Aalto University

School of Science

Degree Programme in Computer Science and Engineering

Visa Korhonen

Availability in Mobile Application in IaaS Cloud

Master's Thesis

Espoo, December 27, 2012

Supervisor: Professor Tomi Männistö

Instructor: Olli Taipale, M.Sc.

Aalto University
School of Science

Aalto University School of Science Degree Programme in Computer Science and Engineering	ABSTRACT OF THE MASTER'S THESIS
Author: Visa Korhonen	
Title: Availability in Mobile Application in IaaS Cloud	
Number of pages: 86 + 13	Date: 27.12.2012 Language: English
Professorship: Software engineering	Code: T-76
Supervisor: Professor Tomi Männistö, D.Sc (Tech.)	
Instructor(s): Olli Taipale, M.Sc. (Tech.)	

Abstract:

Deploying software system into IaaS cloud takes infrastructure out of user's control, which diminishes visibility and changes system administration. Service outages of infrastructure services and other risks to availability have caused concern for early users of cloud. In this thesis existing web application, which is deployed in IaaS cloud, was evaluated for availability. Whole spectrum of different cloud related incidents that compromises provided service was examined. General view from availability point of view of the case Internet service was formed based on interviews. Big cloud service providers have service level agreements effective and long cloud outages are rare events. Cloud service providers build mutually independent domains or zones into infrastructure. Internet availability is largely determinative of users' perceived performance of site. Using multiple cloud service providers is a solution to cloud service unavailability. Case company had discovered requirements for availability and sufficiently prevented threats. Case company was satisfied in cloud services and there is no need to withdraw from cloud. User is a significant threat to the dependability of system, but there are no definite means to prevent user from damaging system. Taking routinely and regularly backups of data outside the cloud is the core activity in IT crisis preparedness. Application architecture was evaluated and found satisfactory. Software system contains managed database service and load balancer as an advanced feature from IaaS provider. Both services give crucial support for the availability of the system. Examined system has conceptually simple stateless recovery.

Keywords: Internet service availability, cloud computing, IaaS cloud, web application

Aalto-yliopisto
Perustieteiden korkeakoulu

Aalto-yliopisto Perustieteiden korkeakoulu tietotekniikan koulutusohjelma	DIPLOMITYÖN TIIVISTELMÄ
Tekijä: Visa Korhonen	
Työn nimi: Availability in Mobile Application in IaaS Cloud	
Sivumäärä: 86 + 13	Päiväys: 27.12.2012 — Julkaisukieli: Englanti
Professuuri: Ohjelmistotuotanto	Professuurikoodi: T-76
Työn valvoja: Professori Tomi Männistö, TkT	
Työn ohjaaja(t): Olli Taipale, DI	

Tiivistelmä:

Ohjelmiston käyttö IaaS –pilvessä saattaa infrastruktuurin käyttäjän kontrollin ulottumattomiin, mikä heikentää näkyvyyttä ja muuttaa järjestelmän hallintaa. Palvelukatkot infrastruktuuripalveluissa ja muut riskit saatavuudelle ovat aiheuttaneet varovaisuutta pilvipalveluiden varhaisissa käyttäjissä. Tässä diplomityössä evaluoitiin olemassa olevan ja IaaS –pilvessä käytettävän web-sovelluksen saatavuutta. Kokonainen kirjo erilaisia pilveen liittyviä tapahtumia, jotka keskeyttävät tarjotun palvelun, tutkittiin. Yleiskuva saatavuuden näkökulmasta katsottuna muodostettiin haastattelujen pohjalta. Suurilla pilvipalveluiden tarjoajilla on voimassa olevat palvelutasosopimukset ja pitkät palvelukatkot ovat harvinaisia tapahtumia. Pilvipalveluiden tarjoajat rakentavat infrastruktuuriin toisistaan riippumattomasti toimivia alueita. Suurelta osalta määräävä tekijä käyttäjien kokeman sivuston suorituskyvyn kannalta on Internetin kautta palveluun liittymisen saatavuus. Useamman pilvipalvelun tarjoajan käyttäminen on ratkaisu pilvipalvelun saatavuuteen. Case-yritys oli löytänyt vaatimukset saatavuudelle ja riittävällä tavalla estänyt riskien toteutumisen. Case-yritys oli tyytyväinen pilvipalveluihin ja pilvestä pois vetäytymiselle ei ole tarvetta. Käyttäjä on merkittävä riski järjestelmän luotettavuudelle, mutta ei ole varmoja tapoja estää käyttäjää vahingoittamasta järjestelmää. Keskeinen toiminto tietotekniseen kriisiin varautumisessa on rutiininomainen ja säännöllinen varmuuskopioiden teko. Sovelluksen arkkitehtuuria evaluoitiin ja se havaittiin tarpeita vastaavaksi. Ohjelmistojärjestelmä sisältää palveluntarjoajan ylläpitämän tietokantapalvelun ja web-palvelimien tietoliikenteen kuorman tasaajan IaaS –palvelun edistyneinä ominaisuuksina. Molemmat palvelut tukevat ratkaisevasti järjestelmän saatavuutta. Tarkastellussa järjestelmässä on käsitteellisesti yksinkertainen tilaton järjestelmän palautuminen.

Asiasanat: Internet-palvelun saatavuus, pilvilaskenta, IaaS –pilvi, web-sovellus

Acknowledgements

I would like to thank Esa Tikkala and Janne Lukkari for making this work possible. Also I want to thank Tomi Männistö for guidance and inspiration and Olli Taipale for support and numerous corrections.

Contents

List of figures

Abbreviations and acronyms

2PC	Two Phase Commit Protocol
ACID	Atomicity, Consistency, Isolation, Durability
AMI	Amazon Machine Image
API	Application Programming Interface
AWS	Amazon Web Services
AZ	Availability Zone
CAP	Consistency, Availability, Partition tolerance
DB	Database
DNS	Domain Name System
EBS	Amazon Elastic Block Store
EC2	Amazon Elastic Compute Cloud
ECU	Amazon EC2 Compute Unit
ELB	Elastic Load Balancing
GFS	Google File System
GPU	Graphics Processing Unit (general purpose)
HTML	Hypertext markup language
http	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IaaS	Infrastructure as a Service
IDE	Integrated Development Environment
IP	Internet Protocol
Java EE	Java Enterprise Edition
LAN	Local Area Network

MVC	Model-View-Controller
NoSQL	Non-relational Database
OASIS	Organization for the Advancement of Structured Information Standards
PaaS	Platform as a Service
RDBMS	Relational Database Management System
RDS	Amazon Relational Database Service
REST	Representational State Transfer
RPC	Remote procedure call
S3	Amazon Simple Storage Service
SaaS	Software as a Service
SAN	Storage Area Network
SDK	Software Development Kit
SLA	Service Level Agreement
SOA	Service-Oriented Architecture
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
SQS	Amazon Simple Queue Service
SSH	Secure Shell
UPS	Uninterruptable Power
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
VM	Virtual Machine
XML	Extensible Markup Language

Chapter 1

Introduction

1.1 Background

Cloud services are currently growing rapidly in IT industry. There are several customer needs that drive this change in industry. Mobile communication and prevalence of networked laptops, smart phones and tablets create need for multichannel access to applications. Also cloud computing has lucrative business model, that benefits customers with cost savings and agility in making business.

Despite big promises, that cloud computing gives, many companies consider that they are not well prepared to move into cloud. Several risks are connected to cloud computing and availability of services is one of them. Major cloud infrastructure service providers have experienced noticeable outages of service. This even has made some, rightfully or not, to become disillusioned about cloud. Cloud computing promises virtualized computing resources that appear limitless. Realistically utilization of infinite resources is illusion. Also there are legal and other concerns that limit transfer of data in cloud outside of certain geographical areas. This geo-location issue limits available possibilities in cloud computing and also further increases concerns about availability of services.

Different IaaS clouds differ substantially. Cloud computing has been a driver for some technical advancement in information technology. For an application architect and web architect cloud calls for rethinking of architectural aspects. Systematic approach to cloud engineering and cloud application architecture will avoid the pitfalls of ad hoc approach.

1.2 Research problem

Deploying software into IaaS cloud and using software services that accompany IaaS cloud makes external services part of the system architecture. Any non-functional feature of architecture need to be considered as always, but architectural features build up differently in cloud. User of public cloud has no control over infrastructure and no visibility into its inner workings. Opaqueness of cloud only increases frustration in case something unwanted happens while using cloud services. Non-functional properties can be unsuitable for some planned system and cause blocking problem for its realization. Although different non-functional properties of cloud cannot be expected to be independent of each other, research concentrates on availability. Therefore the main research question is as follows:

- What compelling requirements for availability and failure and disaster recovery application in cloud has and how architecture in cloud meets them?

In order to clarify and lay ground to the study additional research questions are derived as follows:

- How different services and artifacts in cloud support availability of software system?
- How cloud application architecture support availability of software system?
- What kind of plans of failure and disaster recovery can be done that support business continuation and how to support those plans in cloud?
- What requirements for availability of examined system have been discovered and are those requirements met in examined system?

1.3 Objectives of the study

The main purpose of this study is to evaluate existing web application, which is deployed in IaaS cloud, for availability. Evaluation is preceded by collecting requirements for availability and failure and disaster recovery in IaaS cloud context. Collected requirements are compared to existing and later also coming, application architectures in cloud. Objective is to find, which requirements have been fulfilled and how they have been fulfilled. Goal is also to find possibilities for improvement of availability and, if found, compare those possibilities to current architectural solutions. Found information concerning requirements, cloud services and architecture in cloud is

used in future R&D of cloud applications in company, which developed the case solution.

This study also has a purpose of contributing into the knowledge building process of this particular field of cloud engineering by providing implications for further research. This is achieved through case study combined with exploratory literature review of the common body of knowledge about cloud availability and architecture.

1.4 Scope of the study

The study is demarcated to one application in case company and its particular requirements, architecture and its past success or failure. Central viewpoint is limited in cloud service availability, outages in cloud service and any cloud related incident that compromises service, which application provides. Examined threats include expected cloud related failures as well as any large scale event or unexpected disaster.

As cloud is currently emerging field, given definitions of cloud differ from people to people. In some cases cloud means virtualized data center, which resides in own premises of an organization. In some cases cloud means infrastructure, which an organization uses as a service. Crucial distinction between these infrastructures is the amount of control, that user has over them. Organization, that owns all infrastructures, which makes up the cloud, has complete visibility and control over cloud platform and every aspect that it consists, hardware, software and networking gear. This study takes the viewpoint of user of commercial infrastructure services, because it is the real situation of the case system. Second viewpoint, which this study takes, is looking at the inner workings of IaaS cloud, but it is only secondary and brings more insight into the primary point of view. This order of priorities is hallmark of the scope of this study. User of commercial IaaS cloud has limited options for cloud service provider and limited possibilities to collect evidence about inner working of commercial cloud.

Recovery of cloud-deployed system from failure or outage is examined in context of availability. Mechanisms of recovery and technical tools for crisis management are examined in the extent, that they are related to cloud service outage or other cloud related incident. Among suggested solutions for high availability can be included for example deployment planning, application architecture, software appliances and service agreements.

Cloud application has besides availability also several other non-functional qualities, including cost. Other architectural qualities besides availability are examined primarily how they are connected to availability.

Examined case application is deployed in Amazon cloud. Scope of the study is limited to services in IaaS cloud and their utilization and particularly Amazon Web Services (AWS).

1.5 Methodology

Method in study is case analysis of case company. Context for analysis is application, which has been developed by case company and which is in production at the time of execution of the study. Also literature review method is used for discovering existing knowledge in the topic area and to build theoretical foundation for the case study.

Yin R.K. [1] describes the method of case research. He explains that first and most important condition for choosing research strategy is the identified type of research question, which is asked in research. Typically when case study is chosen, research question primarily asks "how" or "why". Case study is especially used, when boundaries between phenomenon and context are not clearly defined. Case study method allows retaining the holistic and meaningful characteristics of real-life events. Also case study as a research strategy is chosen based on amount of control, that researcher has over events being examined. Case study is favoured, when there is little control over events. Experiment on the other hand requires control over events being examined and isolation of variables being examined. Case study examines contemporary events, which adds two possible sources of empirical evidence into repertoire: direct observation of the events being studied and interviews of the persons involved in the events. Direct detailed observations however, are not always included in case studies. Yin R.K. emphasizes, that case study by no means equals qualitative methods. Instead, case studies can be based on any mix of quantitative and qualitative empirical evidence.

Yin [1] gives technical definition of case study method. Besides scope, the characteristics of a case study include data collection and data analysis strategies.

The scope of a case study:

A case study is an empirical enquiry that investigates a contemporary phenomenon

within its real-life context, especially when the boundaries between phenomenon and context are not clearly evident.

The data collection and analysis strategies of a case study:

The case study enquiry copes with the technically distinctive situation in which there will be many more variables of interest than data points, and as one result relies on multiple sources of evidence, with data needing to converge in a triangulating fashion, and as another result benefits from the prior development of theoretical propositions to guide data collection and analysis.

Some case studies have explanatory function. Those studies have propositions that explain the course of events. Besides explanatory case studies, there are descriptive case studies, which describe course of events and discover key phenomena in them.

Unit of analysis in this case study is one web application that has been deployed to IaaS cloud. Context of analysis is business of service company, which is enabled by the information system. Motivation for selecting this particular system is convenience and the fact, that the system experienced noticeable outage on 2011.

This study examines architectural artifacts in the case system, many of which are services managed by commercial cloud service provider. Study also examines requirements for the system that stem from the business of the company, which owns the system. It also examines disaster planning and preparedness in case company and IT requirements, that stem from them. Cloud provider, which hosts the case system, is examined more broadly than just hosting this particular system. More services of the cloud provider are examined and possibilities that they might provide related to research objective are discovered. Dependability of commercial cloud providers is examined and particular emphasis is on the cloud provider, which case company uses.

Interviews of two different involved persons are used as sources. One interviewed person is system architect and another is technology manager of the case company. Data sources for system architecture are architecture documentation and interviews of the system architect. Documentation of cloud provider and literature are data sources for cloud services. Several online sources are used as sources of knowledge of cloud service providers and particularly archives of cloud provider of case system. Key sources of information in the study are interviews of technology manager in case company. Interviewing the technology manager is conducted in email. Interviews

provide information about several things in case company like business goals and requirements, legal agreements, customer satisfaction and crisis management planning. They provide also information about system requirements and experiences of administering the system after it went live. Experiences of administering the system are particularly important sources, because they provide information about outages and failures and other issues that are related to dependability. One semi-structured interview of technology manager is conducted. Semi-structured interview has thematic questions. Each theme is covered with short question and continued with open question. Question form is attached in appendix A.

Chapter 2

Cloud computing

2.1 Characteristics

American National Institute of Standards and Technology (NIST) [2] defines Cloud Computing as follows:

Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.

There are many models of cloud, which makes creating good definition of cloud difficult. Mostly cloud refers to commercial services, but there are also private clouds, that are provided with own resources and are not from external service provider. General definition of cloud, like NIST gave it, is solely technical. It is without features related to organizational boundaries or economy. Cloud in general sense is not defined as service. In reality, cloud for most users is service provided publicly by external service provider. General definition of cloud is more trivial than reality. It rather reflects only what technically is common to all different clouds. Cloud, that this study is about, is based on commercial services.

Cloud services are used on-demand. User pays for only what he uses and resources can be quickly provisioned and released. There are no entry costs. While mostly cloud works as pay-as-you-go, many consumers use cloud services in free-tier without paying anything at all. Cloud services are affordable for most users, because of economy of scale. Because there are no upfront costs and user is not tied to any contract, available resources appear limitless. Used resources adapt to changing needs. Cloud resources can

be provisioned and scaled faster than any other computing resource. From this usage, that can be scaled both upwards and downwards, comes fluffiness of resource usage that justifies the word "cloud".

Cloud is multitenant. Big service providers service large amount of customers based on self-service. Elasticity and scaling can typically be managed through API, but scaling is sometimes also automated. Besides self-service interface, managing used cloud resources does not need any interaction between customer and service provider. Cloud computing has many different models and the extent that user has control over resources varies. Typically provisioned resources can be configured, but offered services have no customer specific functionality or features. Higher level services give least control to user.

Cloud is accessed through network. Clients that are used with cloud are the same that would be used for any networked resource. This includes web browsers, fat clients, smart phones and tablets mostly. Because cloud is accessed through network, cloud resources appear indifferent of physical location. In some cases however user has the possibility of limiting location in some broad sense like defining continent of geo-location for cloud resource.

Cloud services are naturally opaque from user's perspective. Mechanisms of multitenancy like virtualization and access through network hide every aspect of physical resource. Service providers however deliberately bring visibility to use of resources. Visibility is offered with many kinds of monitoring and metrics. Also because fees are based on actual use, use of resources need to be measured.

2.2 Service models

There are several cloud service models that have different types of service offerings. Services might have higher abstraction level and higher added value for the customer or they can be lower lever services. Common typology between cloud services is as follows:

Cloud Software as a Service (SaaS)

Offered computing resource to use is software. Cloud software is used with client that can be fat client, web browser, smart phone etc. This service model offers alternative to licencing and deploying software. User has no control over used software other than

user specific settings, configurations and data. Neither does user have control over environment, where software is executed. On the other hand software versions, licences, updates and any other management and maintenance are handled by service provider.

Cloud Platform as a Service (PaaS)

Offered computing resource to use is cloud software platform. PaaS is currently the one of the major service models that has the smallest market share. Customer is offered cloud software platform, where he can deploy software of his own choice. Platform in PaaS has abstract nature instead of consisting of machines and other infrastructure, where software typically is executed. Like any cloud, PaaS is scalable and scaling of an application is managed by platform. Cloud platform is restricted environment, that forces user to use particular programming model and proprietary APIs. User of cloud software platform can control other aspects of deployed software. Infrastructure underlying platform cannot be controlled by user.

Cloud Infrastructure as a Service (IaaS)

Offered computing resource to use is cloud infrastructure. Infrastructure comprises virtual machines with computing power, network bandwidth or data transfer and storage. Cloud infrastructure in IaaS is virtualized and there is a possibility, that user is allowed to choose between available operating systems, that virtual machines are executing. Certain IaaS service provider gives user control over whole Linux operating system machine image in virtual machine. IaaS is the service model that gives most control to user. User has complete control over virtual instances from operating system upwards. A lot of IaaS usage is based on open-source software.

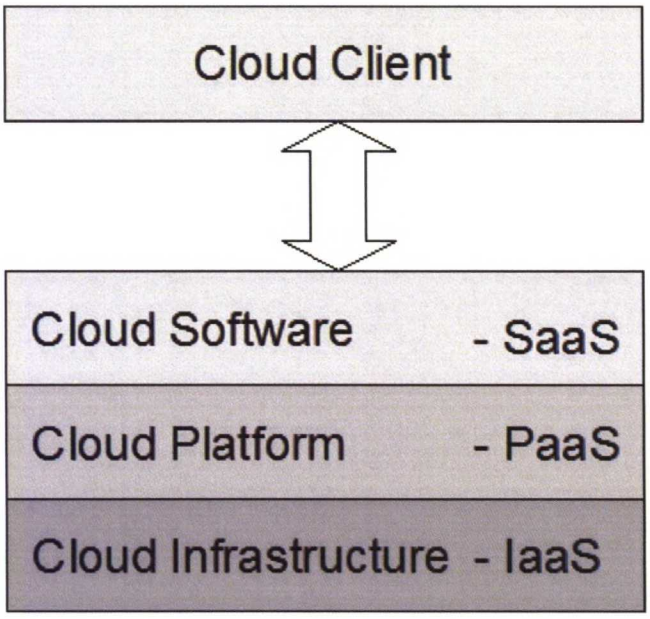

Figure 1, Cloud service models

Division to IaaS, PaaS and SaaS is thorough, because it creates layered model of cloud computing. Each layer is built on top of another. This is not only logical model, but it also many times corresponds to business ecosystem of cloud services. There are several service providers that offer SaaS, which executes on public IaaS cloud. Cloud infrastructure is provided by separate company and SaaS providers have their own distinct businesses of added value services.

Anyone using PaaS or IaaS services is responsible themselves to acquire, deploy and manage software that makes working application in cloud. Distinction between these service models is however significantly blurred. Several IaaS offerings for example have some features of PaaS or SaaS [3].

2.3 Deployment models

There are several deployment models of cloud computing. Two extreme ends of those models are *public cloud* and *private cloud*. Public cloud is made available to the general public in a pay-as-you-go manner. Private cloud on the other hand is cloud that is internal to an organization and not available to the general public. Private cloud may exist in company's own data centre or it may exist in premises of third party. Cloud is private, when it is solely provided for one organization. Two combinations of previous models are *hybrid cloud* and *community cloud* [2]. Hybrid cloud is a model that combines clouds. Exact definition of hybrid cloud is that two or more clouds, that are

separate entities, are integrated together, so that data and applications are portable between clouds. Common usage or the term is however, that it refers to combining private cloud and public cloud. Community cloud is deployment model, that is sometimes also mentioned and it refers to cloud that is provided for a group of organizations and not to more general public.

Private cloud can be made technically similar to public cloud, but they differ in overall size. This is currently true for all companies except such companies as Google or Microsoft, which have large enough data centres. Economy of scale make significant economical difference in public cloud, since they are operated in extremely large-scale commodity-computer data centres. This scale has made possible the factors of 5 to 7 decrease in cost of electricity, network bandwidth, operations, software and hardware [3]. Considering middle size data centres as cloud computing will lead to exaggerated claims for private clouds [3]. Totally irrespective whether claims for private cloud are exaggerated or not, many organizations utilize private or hybrid clouds and still many more are planning or probing for possibilities of using them. Completely private cloud calls for big size, but for small and midsize companies to even completely outsource their data center infrastructure is a solution. Bigger companies on the other hand are able to balance peak loads with hybrid cloud capacity [4].

The Open Group made cloud computing survey in February and March of 2011 [5]. Organizations, that were surveyed were global organizations ranging in size from fewer than 200 to more than 5000 employees.

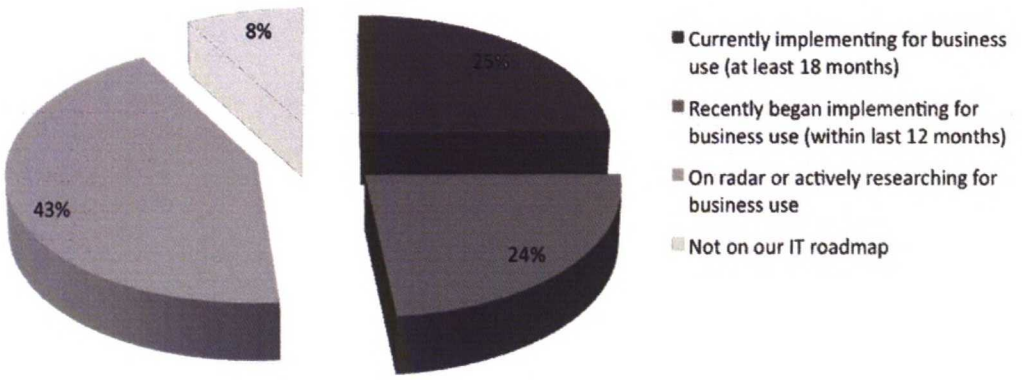

Figure 2, Cloud computing in organization's IT road-map

Approximately half of the organizations had begun to implement business use of cloud

computing. When asked, what deployment model meets best organization's business requirements, hybrid cloud had largest portion of all deployment models.

Hybrid	45 %
Private	29 %
Public	17 %
Unsure	9 %

The large portion of private cloud must be biased by the large size of organizations that responded to survey. While all organizations where global, slightly more than half of them had more than 5000 employees.

2.4 Concerns

Armbust et al. [6] depicted the big picture of cloud computing by examining economics of cloud computing and looking at current situation and future trends. They created their own ranked list of obstacles to adoption and growth of cloud computing. In their ranked list of top 10 obstacles availability of service was ranked to the first place followed by data lock-in and data confidentiality. Their view however seems to differ from how industry sees the same question. The view, that availability in cloud is a big obstacle, is divergent. Leavitt [4] reports about survey by IDC, which asked chief information officers and IT executives, what they rate as their main concern related to cloud computing. Almost 75 percent of respondents said they were worried about security. Having their information and critical IT resources outside the firewall made them worried. Open Group received in its own survey responses that were rather consistent with survey by IDC. Open Group [5] made survey in February and March of 2011 for global organizations ranging in size from fewer than 200 to more than 5000 employees. When asked, what is their main concern with using an *outside cloud provider*, concern that respondents referred most was security with 26% share followed by interoperability/portability with 20% share, vendor lock-in with 19% share and performance issues with 17% share. These responses differed from the overall results, where respondents were asked, what are their biggest concerns surrounding the cloud today. This question was about cloud computing in general. Biggest share in responses had security with 18% followed by integration issues with 17 %, governance with 14% and ability to cope with change with 11% share. Responses like governance and ability

to cope with change reflect consideration of overall effect that cloud computing has to organization. Most respondents answered, that they expect cloud computing to significantly impact business processes in organization and most indicated also, that they don't consider their organization to be prepared for these changes. Noticeable about answers is also, that many respondents were thinking, that deployment model, which was most fit to their organization, was either hybrid cloud or private cloud. Answers reflect the fact that respondents in these large organizations did not consider outside cloud providers as very important to their organization.

2.5 Scalable cloud

Software systems need scalability. Not however all software scales equally well. Vertical scaling or scaling up is increasing performance of a single machine. Scaling out is adding more elements. Software needs parallelization in order to scale out. If this is not feasible, cloud may even be bad choice for that particular purpose [7]. Low cost cloud is enabled by large-scale commodity-computer data-centres [6]. Scaling up performance of a single element leads to expensive high-end machine that does not have much of the economic benefits, that cloud computing has. If however application scales out, resources available in cloud are limitless. In a scale up approach, the software architecture does not play a big role: the better the hardware, the faster the application [8]. Cloud is not equally straightforward, because adding more elements adds complexity. There are more machines and more elements in networks, which makes overall system exposed to failure. Aspiring high-availability for the system puts software architecture into important role.

2.6 On-demand availability

Having computing resources available on-demand in a cloud computing system is a matter of dimensioning resources in cloud. Organizations seeking to operate a cloud face a serious dilemma. If there are not sufficient idle resources available during the busiest service times, requests for resources will need to be denied. If organization operates commercial cloud computing service, this means lost revenue. The costs of maintaining a pool of resources, however, are tremendous. Costs of over-provisioning will erode profit margins.

Hacker T. J. [9] discusses on-demand availability and presents a model that can be used to predict the probability of an N node cloud computing service blocking request due to

insufficient capacity during a busy service period. During the busy periods, the cloud computing system must immediately service requested resources from a pool of free resources or deny access, if it lacks available resources. Requests are not put into hold to wait for resources. Hacker models cloud computing workload based on workload traces, that are available and as close to cloud computing workload as possible. Closest available observed workload came from a large grid computing system. Modelling was done by partitioning the workload into resource classes that are atomically reserved units of service. Each resource class contains different size of node partitions. Hacker examined distribution of two variables in the trace data, elapsed time between job requests and holding time of jobs. He assumed based on distributions, that traced load had been generated by a process, which fits into multiclass Erlang loss model. Probabilities of blocking were calculated from the modelled workload and Erlang loss model. Results were that small resource classes with a limited number of nodes have a low probability of blocking. Secondly, resource classes containing not more than $C/4$ nodes, where C is the number of nodes available in the cloud computing system, have a low probability of blocking, and there is limited improvement in the probability of blocking as C increases. If this rule of $C/4$ nodes is applied to warehouse-scale clouds, that commercial clouds are, result is number of nodes that is sufficiently big for any realistic demand.

Hacker [9] also discusses reliability of cloud computing nodes and replacing failed nodes inside user's cluster of nodes with spare nodes. There are two requisites for increasing the reliability by replacing failed nodes with spare nodes. First, user application must detect failure and respond to the failure in a manner that allows it to tolerate failure. Second, cloud computing system must provide hot-spare nodes that user application can request. Commercial cloud service can be assumed to be sufficient pool of resources for fulfilling the requirement for node reliability.

Chapter 3

Dependable computing

3.1 Introduction

Dependability is desired feature of cloud computing also. While utilizing public cloud, service provider is solely responsible for hardware and its management. Real hardware however still fails and cloud is built from commodity hardware. Difference from user's perspective is, that user has no control or visibility over hardware. Service failures are given in cloud, but how do infrastructure services fail? It is best to look at dependability in cloud infrastructure systematically. Because direct observations are not possible, there need to be other means for understanding.

Avizieniz et al. [10] define concepts related to dependable computing. There are other taxonomies of concepts that are comparable to what Avizieniz et al. have reported. It seems however, that report by Avizieniz et al. is synthesis report and it extends several earlier reports.

Besides dependability computing system has other characterizing properties like functionality, performance, security, cost, usability, manageability and adaptability. Those features are left out from analysis.

Dependability is considered as global concept and concepts that are comprised by global concept of dependability are defined. Dependability comprises reliability, availability, safety, integrity and maintainability. Also concepts that are closely related to dependability are defined. Such concepts are the threats to dependability and the means to attain dependability.

3.1.1 System

System is a fundamental concept that needs to be defined before defining other concepts.

A system is an entity that interacts other entities i.e. other systems [10]. System boundary separates enclosed system from its environment. Other systems include hardware, software, humans, and the physical world with its natural phenomena. System has several different attributes:

Function is what the system is intended to do.

Functional specification describes the function of the system.

Behaviour is what the system does to implement its function.

Total state comprises set of following states: computation, communication, stored information, interconnection, and physical condition.

Components are what are comprised by the system. Component can be another system or atomic component, whose further internal structure is not of interest.

Structure bounds components together in order to interact. Structure enables the system to generate its behaviour.

From external perspective system has still further attributes:

Service is delivered to user by the system. Service is a sequence of systems external states, and as such, it is systems behaviour as perceived by the user.

User system receives service from provider system

Service interface is where service delivery happens. Service interface is part of service provider's system boundary

External state is the part of the provider's total state, which is perceivable at service interface.

Internal state is the part of the provider's total state, which is not externally perceivable.

Use interface is the interface of the user at which the user receives service.

3.1.2 Threats to dependability

Service failures

Avizieniz et al. [10] present failure modes of service and causes to failures:

Correct service implements the system function and is delivered to user.

Service failure is when the delivered service deviates from correct service.

Service outage is a period of delivery of incorrect service, including no service at all.

Service restoration is restoring correct service after period of service outage

Error is some part of total state of system, which can lead to its subsequent service failure. Service failure is by definition visible in service interface. The cause of an error is *fault*. This means that service failure, that is perceivable externally to system, is necessarily preceded by error, deviant state of system. Error in turn is necessarily caused by fault, whether it is visible or known or not. The opposite is not true. Error does not necessarily lead to a service failure and fault does not necessarily cause an error. When fault causes error, it is *active*. Otherwise it is *dormant*.

When a system contains a set of several functions, system may suffer *partial failure*. If one or more of the services implementing the functions fail and a subset of needed services still are offered to the user, system is left in a *degraded mode*.

3.1.3 Attributes of dependability

Avizieniz et al. [10] define dependability as the ability to avoid service failures that are more frequent and more severe than accepted. Dependability encompasses the following attributes:

Availability is readiness for correct service.

Reliability is continuity of correct service.

Safety is absence of catastrophic consequences on the user and the environment

Integrity is absence of improper system alterations.

Maintainability is ability to undergo modifications and repairs.

There are several means to attain different aspects of dependability. Those means can be grouped into four major categories:

Fault prevention is comprised of means to prevent the occurrence of faults.

Fault tolerance is comprised of means to avoid service failures in the presence of faults.

Fault removal is comprised of means to reduce the number and severity of faults.

Fault forecasting is comprised of means to estimate the present and the future number of faults and the likely consequences of faults.

Reliability and availability are two concepts that are close to each other. Reliability of a system is defined as continuity of correct service and measured as continuous timespan of correct service without failure. Availability on the other hand is the proportion of time that the system is in working order or up. Looking availability closer, however, reveals that availability is dependent on repairability. For a repairable system availability is defined as [11]:

Availability=MTBF/(MTBF+MTTR)

where

MTBF=Mean Time Between Failures (reliability)

MTTR=Mean Time to Repair (maintainability)

Software systems are notoriously elastic and almost always repairable. Failure that is not repairable could be some irreversible action, launching a missile for example. In typical case it is equally beneficial to improve repairability of a system than improve reliability of a system.

3.2 Errors

3.2.1 Error propagation

Creation and propagation of errors is shown in figure 3. Using concepts previously defined, fault is causing an error in system A, and hence fault is *active fault*. Error causes another error inside same system. Error propagation within the component is caused by the computation process: an error is successively transformed into other error.

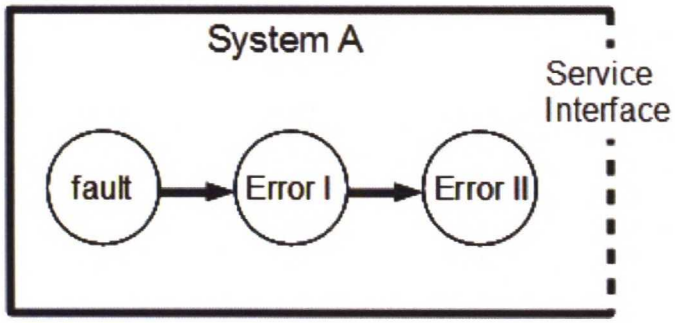

Figure 3, error propagation

Noticeable in error propagation presented in figure 3 is, that both errors are in internal state of the system. Because neither error is perceivable outside of the system, there is no deviation in service interface and thus no service failure occurs.

Figure 4 presents another error creation. This example shows two distinct components that interact. System A is service provider for system B, which is user for system A. Error propagation is caused by the computation process, but in this case error in system A is in service interface. Because error is perceivable externally, there is service failure in system A.

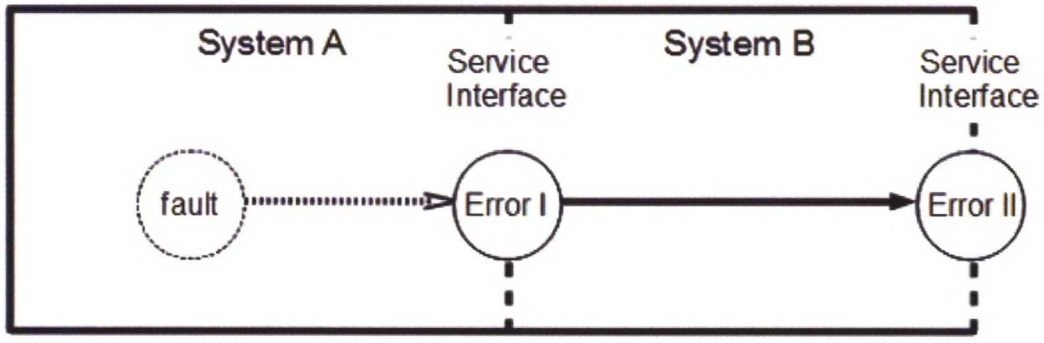

Figure 4, service failure

Failure in service provided by system A is external fault for system B. Fault is outside of system border of system B, but it causes error in system B. Figure 4 shows, that error in system B is in this example also in service interface. It means that there is service failure in system B also. Error has propagated from system A to service provided by system B.

3.2.2 Fail-controlled system

When service fails, there can be different output from service depending on error in case. Implementation of system can however also be such that system fails only in specific mode of failure. System produces only certain type of output, when failure in service occurs. Systems with this kind of implementation are fail-controlled systems [10]. Our particular interest in here is software system that is deployed to cloud infrastructure, Infrastructure-as-a-Service (IaaS). Considering failures in system, that are caused by infrastructure failure, system is *fail-silent*. System, whose failures, to an acceptable extent, produce only silence as output, is fail-silent [10]. Mode of failure in system is important consideration. It needs to be known, or forecasted if it is not known, in order to design mechanisms of service restoration for the system. Dependability specification defines, what kind of failures are acceptable and in what extent failures are acceptable to occur.

Figure 5 presents failure scenario, where failure happens in a controlled fashion. Example system in figure 5 comprises two components that are sub-systems, system A and system B, which of system A is deployed to IaaS cloud infrastructure. System A comprises cloud services from IaaS cloud and application software that is deployed in cloud. Internal state of cloud service is completely undetectable in use interface of cloud service. Underlying infrastructure fails in failure scenario and that causes failure in service interface of system A. Failure mode in question is silence. System A does not respond to any message. Figure 5 shows error propagation from service interface of system A to system B. In this example error in system B is in internal state of system B. Service interface of system B is not affected and there is no service failure in system B. This is due to *fault-tolerance* is system B.

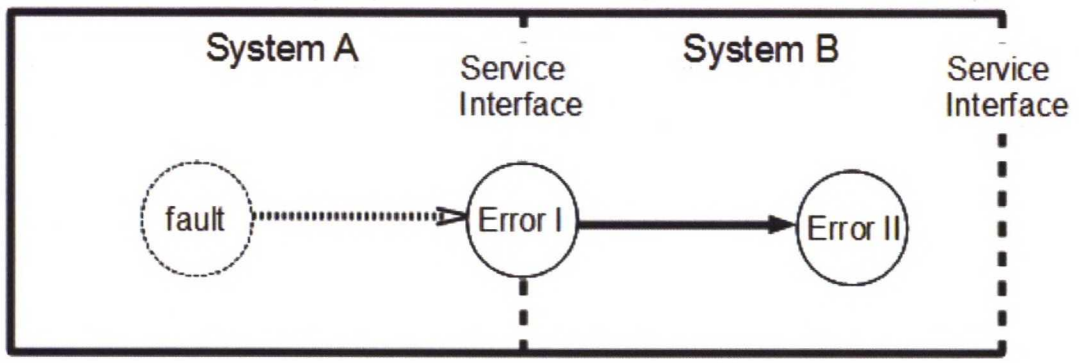

Figure 5, fault-tolerance

3.2 Fault tolerance

The main concern for fault tolerant designs is the ability to continue delivery of services in the presence of faults in the system. In general case fault tolerance is achieved through the use of redundancy in the hardware, software, information or time domain [12]. A method of achieving fault tolerance is to perform multiple computations through multiple channels, either sequentially or concurrently [10]. Except for some very special systems that need to be very dependable, however, majority of software systems have no redundant computing performed concurrently through multiple channels. For such single-version software recovery mechanisms are a few.

Fault tolerance is carried out via error detection and system recovery [10]. Common way to detect error is timing constraints. Watchdog timers are timing checks with general applicability that can be used to monitor for satisfactory behaviour and detect lost or locked out components [12]. Avizieniz et al. [10] present three techniques for recovery, rollback, rollforward and compensation, of which rollback should be tried first. A restart, or backward error recovery, is conceptually simple, general and has the advantage of being independent of the damage caused by a fault. There exist two kinds of restart recovery: static and dynamic [12]. A static restart is based on returning the module to a predetermined state like the initial reset state. Dynamic restart uses dynamically created checkpoints that are snapshots of the state at various points during the execution.

The structure of the system, and especially the nature of any redundancy that exists in it, influences whether service failure will occur in the system or not [10]. Principle of redundant processing nodes is, that there is a group of several processing nodes and any member of the group is capable of supporting the processing functions of any other member [11]. Mechanism for recovery is performing fail-over when one processing node becomes non-responsive.

Redundancy in a system is a mean to achieve fault-tolerance and dependability. There are however two types of redundancy in system, protective redundancy and unintentional redundancy [10]. Protective redundancy is explicitly intended to prevent an error from leading to service failure. Unintentional redundancy is created because redundancy is needed for normal operation, when system behaves satisfactorily. Unintentional redundancy has unexpected result of increasing dependability. These two

categories are not mutually exclusive.

In general case of redundant processing nodes there is redundant n+k configuration, where k processing nodes serve as spares for the n active processing nodes in the system [11]. Simple but important special case of configuration model is *active-standby*, where there is one active node (n=1) and one standby node (k=1). Another similar configuration is active-active, where there are two active nodes (n=2) and no spares (k=0).

Chapter 4

Software architecture in cloud

4.1 Fit frameworks and styles

IaaS cloud can be said to be for general purpose applications. Cloud is built with commodity hardware and examining infrastructure services reveals, that services have actually nothing, that would justify calling IaaS anything else than general purpose computing. Cloud computing is closely linked to some architectures and frameworks, most notably service-oriented architecture (SOA). Allegedly Amazon Web Services was initially created, when Amazon tried to implement strict form of SOA in their internal systems. After building the required infrastructure Amazon decided to offer publicly the same scalable cloud that it uses for its own systems [13]. This kind of link between SOA and cloud is interesting, but SOA in its entirety is a complex concept and it is completely outside the topic of this study. The case architecture, that is examined, does not adhere to SOA, in a way that OASIS group defines SOA [14], and, in general, cloud computing actually does not seem to come with any particular software architecture. There might still be architectures and frameworks that are more fit to cloud than others.

Application that can be made to conform to the framework of grid computing is particularly easy to move into cloud [15]. Characteristic for grid computing is that it is used for large batch processing tasks that need much computational power. Another characteristic is that applications are not interactive. Grid computing is distributed computing, where several computers are loosely coupled in network to solve common problem. Grid programming utilizes resources in distributed network by dividing processing task into several smaller ones, that can be processed parallel independently from each other. Processing smaller tasks parallel solves in short time large problem,

that otherwise would take long time to solve. Fault-tolerance in grid application is achieved by re-running failed task. When problem scales horizontally and fits into programming model of grid computing, it is a good fit into scalable cloud also.

4.2 Web application

There are numerous descriptions of architecture of a web application. Oppenheimer D. [16] examines architectures of Internet services. The research examines particularly dependability and failures of Internet services. Examined cases are however Internet services, which are large-scale. This is typical in descriptions of Internet services and cloud computing. Oppenheimer makes clear, how different large-scale services are. Scaling service into Internet-scale increases complexity with orders of magnitude. This study on the other hand is about small service. Descriptions of large-scale services show only few familiar issues. This study needed to examine basic building blocks of an Internet service.

Web application sets characteristic set of requirements for architecture. Characteristic for web application is that application is interactive. On the other hand typical web application is not particularly computationally intensive. The usability of web application is highly sensitive to user-perceived latency.

Web applications and services have predominant architectural style, which is Representational State Transfer (REST). REST was first introduced by authors of Hypertext Transfer Protocol (HTTP), particularly Fielding R.T., and introduced implementation of REST was no less than World Wide Web, although web's conformity to REST is not total [17]. REST is merely constraining web's architecture by setting constraints for behaviour of its components. REST describes as well, how well-designed web application behaves.

Representational State Transfer involves client and server. Client requests resource from server and server responds to client with representation of resource and its internal state. Web page can present such representation of resource. Whereas representation corresponds to state of the system, client initiates state transfer by requesting resource. Web links can present such state transfers. This makes collection of web pages into virtual state machine. Some web pages are completely static. It is counter-intuitive, that also collection of such static web pages would be a state machine. That however only means, that server does not have internal state and only client has state. REST

specifically states, that state transitions are initiated by client.

An architectural style is a coordinated set of architectural constraints. Constraints restrict architectural elements in roles, features and allowed relationships among them. Set of architectural design decisions that conform to the architectural constraints is named with the style. Also architectural properties that are induced by applying the style are named with the style [17].

REST consists of a set of network-based architectural base styles. Each base style is architectural constraint. These constraints are replicated repository, cacheability, client-server, layered system, statelessness, virtual machine, code on demand, and uniform interface [17]. Many depictions of REST omit replicated repository and virtual machine from the list for the sake of conciseness.

Client-server architectural style is the first of the constraints. Design principle behind client-server is separation of concerns.

Stateless communication is the second constraint. Design principles behind stateless communication are scalability and reliability. Stateless communication between client and server is called client-stateless-server style. Constraint is that each request from client to server must contain all of the information necessary to understand the request. It means that server is unaware of client state outside of single request processing. Exact semantics of state mean a lot for the behaviour of networked system. Client-stateless-server style is about communication between client and server.

Cache constraints improve network efficiency. Cache constraints require that the data within a response to a request be implicitly or explicitly labelled as cacheable or non-cacheable.

Replicated repository is about having more than one copy providing the same service.

Code-on-demand style improves system extensibility. REST allows client to download and execute code in the form of applets or scripts. Extending client by downloading code after deployment reduces complexity of client. REST allows downloading code on demand, but also restricts, that code can be disabled in some contexts.

Virtual machine is an execution environment inside client for server-provided logic. Loading code on demand from server requires execution environment in client for downloaded code.

Uniform interface improves the visibility of interactions. REST restricts component interfaces by applying the principle of generality in component interfaces. This simplifies overall system architecture. Definition of uniform interface is given with four interface constraints. Those constraints are *identification of resources*; *manipulation of resources through representations*; *self-descriptive messages*; and, *hypermedia as the engine of application state.*

> ***Identification of resources*** for example using URIs. The key abstraction of information in REST is a resource. Any information that can be named can be a resource: document, image, temporal service and so on. Resource provides generality by encompassing many sources of information without artificially distinguishing them by type or implementation. REST transfers representations of resources between components instead of resources themselves. A representation consists of data and metadata describing the data. Abstract definition of resource enables author to reference the concept rather than some singular representation of that concept. That allows content negotiation based on characteristics of the request and late binding of the reference to a representation.
>
> ***Manipulation of resources through representations*** provided that client has permission to do so. Client modify or delete the resource on the server through representations, which contain enough information for that including metadata attached.
>
> ***Self-descriptive messages*** including metadata and control data attached. Each representation in a message includes enough information to describe how to process the message. A representation can be processed by the recipient according to the control data of the message and the nature of the media type.
>
> ***Hypermedia as the engine of application state*** manipulates resources. A network of web pages forms a virtual state machine, allowing a user to progress through the application by selecting a link and submitting data-entry forms, with each action resulting in a transition to the next state of the application by transferring a representation of that state to the user.

Uniform interface between components and visibility of interaction are central and the most distinguishing features of the REST architectural style compared to other network-

based styles.

Layered system style constrains component behaviour such that each component can detect only layer with which they are directly interacting. This is despite the visibility of interactions and enables intermediary components.

Notable architectural properties that are induced by applying the REST architectural style is dynamic substitutability of components, and processing of actions by intermediaries.

There is controversy about what kind of application architectures actually complies with REST and what does not. There are also claims, that principles of REST have been frequently misunderstood. REST is not a standard but an architectural style, which makes unambiguity of definitions difficult to achieve. Fielding R. T. [18] discusses for example, how to define resource names and hierarchies. Unless servers instruct clients on how to construct appropriate URIs, such as is done in HTML forms and URI templates, resource names or hierarchies form domain-specific standard. That in turn creates coupling between client and server. Fielding argues that API, that is not hypertext-driven, is not REST compliant (RESTful). Any kind of coupling between client and server makes API not compliant with REST. It is frequent practise to call some API REST, when API is HTTP based and consists of simple and visible queries, even if the API is not hypertext-driven. That kind of API can be used by fat client. Although this naming practise is inaccurate, such design decisions are not due to lack of understanding. Just because networked API is well designed and well behaving, doesn't mean that it is exactly compliant to REST style. Requirement for scalability was behind design of REST and required scalability was set to be internet-scale. Domain specific API, that is HTTP based, can obviously be designed without hypertext. It would indeed be more accurate to call it remote procedure call (RPC) than to call it REST, but calling it REST is nevertheless established practise.

REST is used in case architecture for implementing web services. These web services are the whole interface from the application tier towards the client tier for manipulating resources.

The REST architectural style induces several important architectural properties that support availability of the system. Notable architectural properties are replicated resource providing the same service, dynamic substitutability of components, and

processing of actions by intermediaries. Possibility to replicate resource makes possible to create redundancy into system, which is necessary for building fault-tolerance. Processing of actions by intermediaries makes possible load balancers that divide load between redundant sites and dynamic substitutability of components remove dependency from any single point of failure. These are benefits of web services and they are taken for granted when designing web applications.

4.3 Stateless design

Cloud computing is built on unreliable commodity hardware and design is done with the possibility of failure in mind. Design for failure requires, that virtual instances and processing nodes running on them are considered transient. Such nodes cannot contain data that must persist beyond any application instance [19]. Application should be made as stateless as possible by pushing the state out of the software, separating processing and data as much as possible. Carolan et al. [19] list techniques for making application stateless. State can be pushed down to back-end database. Also state can be pushed out to the user in the form of cookies or state coded into URLs.

JavaServlet technology for example offers possibility for identifying a user across more than one page request or visit to a Web site and to storing information about that user. Tracking session across more than one page request makes stateless HTTP protocol support session state. Servlet container, which is Java EE compliant, includes possibility to use session object through Servlet API for server program to use [20]. User session is uniquely identified with a random number and it is typically put into cookie that is sent in HTTP response into client. Client will store the cookie for the time of browser session. Alternative way to set session id is by URL rewriting. Servlet API allows server program to store object-valued attributes into session. This data is managed by servlet container where it is stored in memory until session expires or it is invalidated. Supposing, that Servlet API is used for storing session data, in-memory session in server becomes soft-state. It is data, that is lost, if server restarts. That is why stateless design requires that data to be saved into database, where it persists beyond any processing node. This leaves server with as little soft-state as possible. If data is not lost in server restart, completely seamless failover becomes possible and from user point of view, there is no discontinuation of service.

4.4 Multi-tier deployment

Deployment of application into cloud requires that hardware is organized into physical layers, so that scaling out is possible. Cloud deployment although means that there are virtual instances and services instead of physical hardware. Common way of partitioning web applications into physical layers is to have client tier, application tier and data tier. Client tier being mostly thin client i.e. web browser on user desktop. Server in the application tier is responsible for hosting web server and application server. Server in data tier is responsible for hosting database server [21].

The Model-View-Controller (MVC) is a well-known design pattern that is a useful way to architect interactive software systems [22]. The key idea behind MVC is to separate user interface from the underlying data represented by the user interface. MVC pattern divides application into layers. Layer is however different concept than tier. A layer is a logical structuring mechanism for the elements that make up software system, whereas tier is physical structuring mechanism for the system infrastructure [23]. Leff et al. notice that MVC design pattern is partition-independent, because it is expressed in terms of an interactive application running in a single address space in which partitioning issues do not arise. They find that it is much harder to apply the MVC design pattern in the web application context, because web applications run on networked architecture and are consequently location dependent. Web browser as a client only renders View, the user interface layer, whereas server generates the View that is rendered on the client. When web application is multichannel and has different types of clients, like mobile clients and web clients, this further adds complexity. This is because clients in smart phones or tablets are often fat clients that are deployed into client devices, thus they are of different type than thin clients.

Chapter 5

Shared data in cloud

5.1 Distributed database

Distributed database systems are used generally for increased availability, performance and scalability [24]. These advantages apply also in cloud and distributed database is naturally fit in networked architecture in cloud. *Distributed database* has been defined as a collection of multiple, logically interrelated databases distributed over a computer network [24][25]. *Distributed database management system* has been defined as the software system that permits the management of the distributed database and makes the distribution transparent to the users [24][25]. Latter definition clarifies significantly the first. Logical connectedness as a characteristic of a distributed database delimits many constructs, like a collection of files, outside of the category. Some of the characteristics of distributed database are that it is structured and accessed via a common interface [25]. Communication between components of distributed database is done over a network. This definition rules out databases that reside in one computer. Also it rules out computing in multiprocessor systems that lack the overhead of communication over a network. It is not parallel computing in database system that defines distributed database. Database management systems in multiprocessor systems are termed *parallel database management systems*, which is a distinct term [24][25].

Transparency in different levels is important characteristic of distributed database. *Distribution transparency* was already mentioned in the definition of distributed database management system. Distribution transparency comprises location transparency and naming transparency. *Location transparency* refers to the condition that the command used to perform a task is independent of the location of data and the

location of the system where the command was issued [24]. *Naming transparency* refers to the condition that once names for objects have been specified, the named objects can be accessed unambiguously without additional specification [24]. Copies of data may be stored at multiple sites for better availability, performance and durability. *Replication transparency* refers to condition, that user is unaware of the existence of copies [25]. Data may also be fragmented into multiple sites. Distributed fragments may be vertical fragments or horizontal fragments. Horizontal fragments are sets of tuples (rows) whereas vertical fragments are subsets of columns of the original relation and both types can exist at the same time. *Fragmentation transparency* refers to condition, that user is unaware of the existence of fragments [24]. Replication and fragmentation of data may exist at the same time and they are irrespective of each other.

5.2 Concurrency control

Applications, that use database systems, are typically multi-user systems. This is easy to forget, when users are purposely unaware of each other. Web applications especially can have large amount of concurrent users. Multiprogramming operating systems, that allow the computer to execute multiple processes at the same time, are self-evident in modern systems, but networked architecture causes such problems, that concurrency once more becomes important factor. Concurrency control is essential part of every database management system irrespective whether database is distributed or centralized.

Transactions are used in database management systems in order to achieve correct results and to keep database in consistent state. Transaction is a logical unit of database processing that includes one or more database access operations. Operations, that access database can read, update, insert or delete data in database. Retrieving data from database is different from any other type of operation because it does not change the state of data. Handling transaction, that is read-only, containing only read operations, is more straightforward. An ideal transaction has been characterized with ACID properties. These properties are as follows [24]:

Atomicity

Transaction is an atomic unit of processing, which is either performed in its entirety or not performed at all.

Consistency

Transaction preserves consistency of database by transferring database from one consistent state to another consistent state.

Isolation

Transaction should appear to be isolated from other transactions. Transaction should not be interfered by any other concurrent transaction.

Durability

Changes in database made by committed transaction must persist in the database.

Transactions need to be executed concurrently, so that operations in transaction are interleaved with operations in other transactions. Interactive multi-user application cannot be implemented satisfactorily with database management system that would execute transactions successively. That kind of system would serve one user at a time, during which others would be waiting. It would waist most of the computing resources, which would remain idle most of the time. Concurrent transactions are inevitable and they need controlling in order to achieve isolation between different transactions. Problems arise, when more than one transactions access same database item and execute conflicting operations.

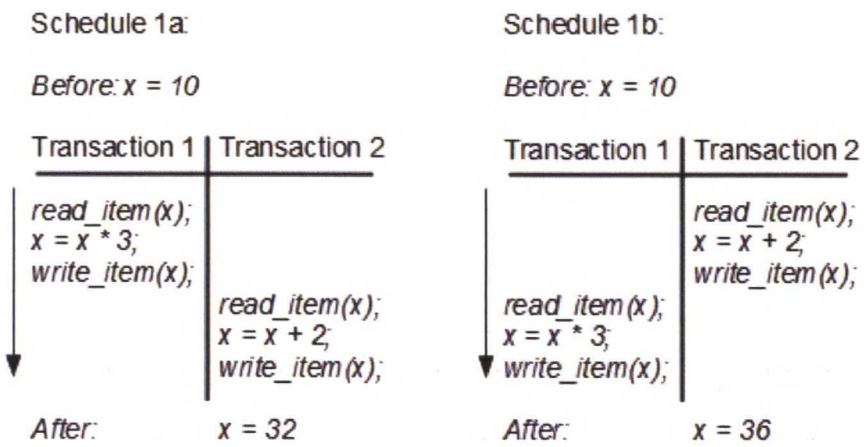

Figure 6, transactions accessing same database item

Figure 6 shows two transactions, that access the same database item marked with symbol x. Both transactions are executed approximately at same time. Transaction 1 is executed in its entirety before execution of transaction 2 in schedule 1a and in schedule

1b the order of execution is the opposite. Both transactions in this example contain simple arithmetic. Figure 6 shows also results from executing either schedule. Results differ depending on which transaction was executed first. Despite the fact, that results are different, both results are correct, at least from transaction point of view. Transactions are isolated from each other and there is no violation from an interfering transaction that would lead into incorrect result.

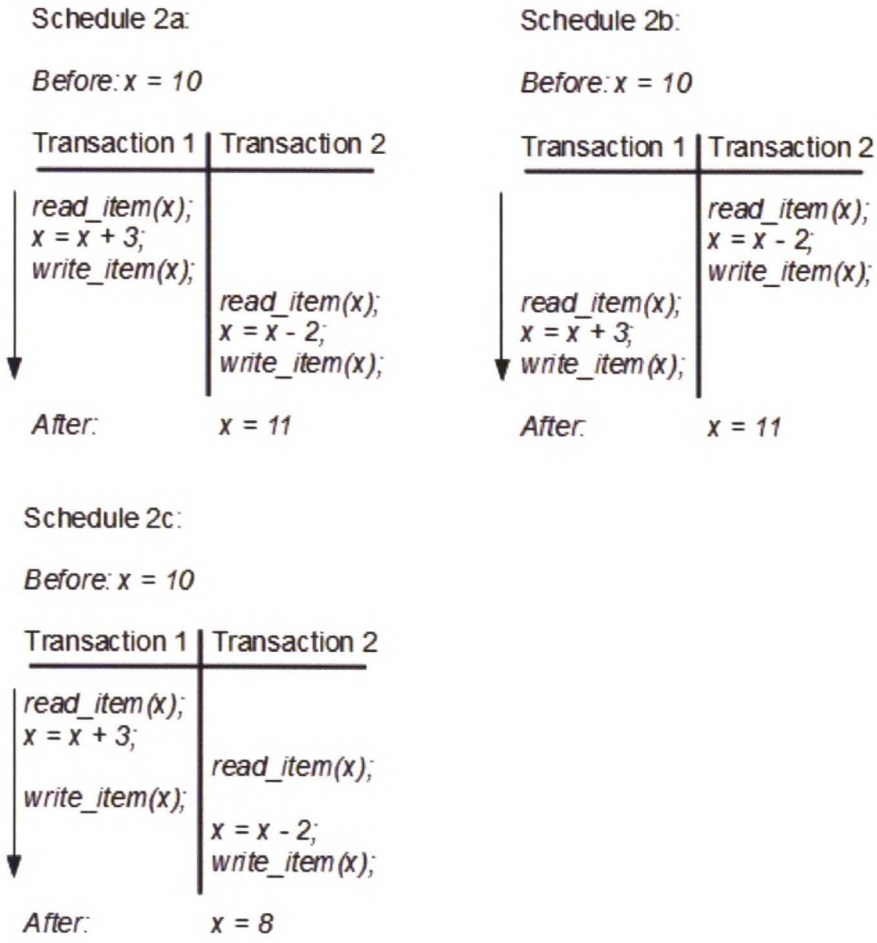

Figure 7, lost update violation

Figure 7 shows another example, where there is a violation in schedule 2c. First two schedules 2a and 2b do not contain violation. Transactions in those schedules are executed serially, one after another. Transaction 1 is executed in its entirety before execution of transaction 2 in schedule 2a and in schedule 2b the order of execution is the opposite. Those two schedules show equal result. Schedule 2c, which contains lost update violation, produces incorrect result. Read operation in transaction 2 reads initial

value of item 'x' in schedule 2c and write operation in transaction 2 over-writes the value written by transaction 1 causing lost update violation.

Transaction is a unit of execution, that ends with commit operation or rollback, which guarantees atomicity with serial transactions. It is noteworthy that, while commit operation is started by user, rollback can be started either by user or database management system. Transactions need to be executed concurrently so that operations in transactions become interleaved and operating system divides processing time between several processes in unpredictable way. On the other hand large numbers of possible schedules, where operations are interleaved, produce incorrect result. This makes concurrency control in database management system essential. Database management systems implement concurrency control with complex protocols that produce different levels of isolation. There are several techniques that are used in concurrency control protocols, and most common is locking. Types of protocols that use other techniques are timestamp protocols, multiversion protocols and certification protocols. Each implementation of concurrency control protocol has its particular characteristics. Even when protocols produce the same level of non-interference, they still differ in other characteristics like latencies that they cause, frequency of occurrences of rollbacks, possibility of deadlock or starvation and so on. Distributed database sets its own requirements for concurrency control protocol, because it is interconnected with network, which contains the possibility of failure.

5.3 Database replication

Replicating database provides increased availability by removing single points of failure. Consequently, even when some sites are down, data bay be accessible from other sites [25]. Replication may in some cases also have the benefit of distributing load between replicated sites. This in turn may improve performance and scalability. Replicated databases are however diverse and they come in all kinds of configurations. In order to understand particular replicated database it is necessary to know the factors of replication.

Outstanding factor of replication protocol is where updates are performed. There are basically two kinds of techniques, single-master technique, which is also called centralized, and multi-master or distributed technique [25]. Single-master technique performs updates first on a master copy, from where updates are then propagated to

other copies, which are called slaves. Multi-master or distributed allows updates over any replica. One of the names of multi-master technique is update-anywhere, which is rather descriptive name.

Essential factor of replication protocol is how updates are propagated to the other copies. The alternatives for update propagation are eager technique and lazy technique. Eager techniques perform the updates of other copies within the context of the transaction that has initiated the write operations [25]. Transaction that accesses replicated data items have to be executed at multiple sites and thus it is a global transaction. In lazy update propagation the transaction commits as soon as one replica is updated. The updates propagate to other copies sometime after the initiating transaction has committed. Propagating updates to other copies after transaction has committed is referred to as refresh transaction.

When a global transaction updates all copies of a data item at all replicated sites, the values of these copies may be different at a given point in time. Equality of all copies of replicated item is referred to as mutual consistency. This extends the concept of consistency of centralized database. Global transaction has both transaction consistency and mutual consistency [25]. Tightly synchronized replicas in distributed database create strong mutual consistency. Because distributed database runs on networked environment, strong mutual consistency and synchronous transactions require the execution of two phase commit protocol (2PC) at the commit point [25]. On the other hand relaxed synchronization of replicas creates weak mutual consistency. Weakly mutually consistent replica values may diverge over time, but will eventually converge. This is referred to as eventual consistency [25].

Replication transparency is a desired characteristic of replication protocol. Transparent replication allows referring to data items as logical items instead of physical copies of data. Sometimes replication transparency might be referred to as "a dimension" of replication technique.

5.4 Feasibility of distributed database

World Wide Web and, more recently, cloud computing have increased interest in distributed and parallel data management. Much focus has been on feasibility of distributed database that has desired properties. Gilbert et al. discuss feasibility of distributed database that has particular set of guaranteed features, consistency,

availability and partition-tolerance [26]. Partition tolerance is desired feature in real world networked system. Gilbert et al. prove that such system, that would have all those properties at same time, is not possible. Prove consists of assumption of algorithm, that has all the stated properties. Properties are formalized and presumed to be in assumed algorithm. They construct an execution of assumed algorithm and show that contradiction follows from execution. Construct is a prove by contradiction and shows, that it is not possible to guarantee consistency, availability and partition tolerance in distributed data management algorithm [26]. Gilbert et al. discuss, that while it is impossible to provide all three properties, it is possible to achieve any two of those three properties. Possible solutions that have two of the three properties are:

Consistent, Partition tolerant

Liveness of the system is guaranteed only when there are no failures. If failures occur, liveness of the system is weakened or lost.

Consistent, Available

It is possible to presume, that there are no partitions, when data is inside one system. System runs on LAN for example.

Available, Partition tolerant

It is possible to provide weakened consistency in an available, partition tolerant setting.

Gilbert et al. conclude, that many real-world systems settle with returning "Most of the data, most of the time". Both the impossibility result and the possible solutions have become known as CAP-theorem (Consistency, Availability, Partition tolerance).

5.5 Eventual consistency

Mostly cloud computing has made known eventually consistent databases. Certain types

of distributed databases used in cloud are called eventually consistent. They are called eventually consistent because of a lag in data replication that is visible to users, who are using database concurrently. When one user have updated some particular value in database, another concurrent user might still read the old value of same database item after the update by the first user had already ended. If this was the only exceptional feature of that database, it would be enough to compromise the consistency of the database.

Database system can achieve data consistency, and often does so by using transaction protocols. That kind of system is not, however, tolerant to network partitions. They fail as a whole under certain scenarios. In larger distributed systems network partitions are a given [27]. This means there are two choices on what to drop, either consistency or availability, since they cannot be achieved at the same time. Distributed database that is eventually consistent uses different concurrency control than more conventional centralized database. This concurrency control is based on voting and it makes availability possible for concurrent users even in distributed database in networked architecture. Fundamental units of voting in concurrency protocol are so called quorums, minimum numbers of nodes available to operation. Supposing, that write is executed in database, during which network partition exists, write is acknowledged by those nodes that respond and are in same network partition. Update is written to those available nodes synchronously inside successful transaction. All remaining nodes are updated with lazy mechanism after transaction and actual writing has ended. Lazy update happens after network partition has ended and previously unavailable nodes have become available. Quorum defines minimum number of acknowledged nodes for operation. In a situation, where database item is updated and nodes that are part of transaction do not include all nodes, only those nodes, that take part of transaction, have their data write-locked. This scenario is possible in case of network partition and, in case of some cloud databases, also when the value of write quorum has been configured to one.

Consistency is not very straightforward thing, because there are two ways of looking at consistency. One is from the client point of view: how client observe data updates. The other is from the server side: how updates flow through the system and what guarantees systems can give with respect to updates. W. Vogels [27] discusses conditions, where consistency is eventually achieved with voting protocol in scalable distributed database

in the presence of failures. Both write and read availability of database can be guaranteed when multiple versions of data are allowed to exist in database at same time. This requires, that differing versions of data that form during network partition, are reconciled after network partition ends. Consensus protocol requires that sequence of updates is known and there are several different techniques to implement it. During network partition all nodes cannot be updated, but after network partition ends, updates eventually flow through the system and all nodes are updated. Supposing that W denotes write quorum R denotes read quorum and N denotes total number of nodes, configuration, that satisfies condition $W+R > N$, guarantees strong consistency. This is because groups of nodes in write and read operations are guaranteed to overlap. Quorum means also, that if condition for the number of acknowledged nodes is not met, operation fails. Failure scenario, where quorum is not met, means losing liveness of the system. Eventual, weak consistency arises, when $R+W <= N$. In case the unavailability of application is considered unacceptable, weaker consistency is configured. If $W < (N+1)/2$, write quorum is no more than half of total nodes, there is a possibility of conflicting writes. In that case durability of update is not guaranteed. Scenario, that leads into incorrect result of concurrent updates, is that write quorum is for example one. During update transaction only one replica of data is write-locked. Networked architecture causes lag and there might even be network partition. Another concurrent user might do another update into another database component after first update already finished, but before the update propagates into that other database component. This is because concurrency control system in distributed database does not lock the data item in second database component. In this scenario there are two successful transactions, that are conflicting and the end result is, that other update is lost. It means that data durability is compromised after successful write transaction.

Eventual consistency is found also from modern relational database management systems (RDBMS) that use more conventional concurrency control protocol, even when they have not been scaled into any particularly large scale. Conventional primary-backup replication is enough to create eventual, weak consistency. RDBMS might provide replication techniques either in synchronous or asynchronous modes or both. In asynchronous mode the updates arrive at the backup in a delayed manner, often through log shipping. That means that updates do not arrive inside transaction like in synchronous mode. This delay represents inconsistency window, if read from backup

node is allowed. If the primary replica fails before the logs are shipped, reading from the backup replica will produce old, inconsistent reads.

Chapter 6

Amazon Web Services

6.1 Amazon cloud infrastructure

Amazon is the undisputed IaaS revenue leader with its Amazon Web Services (AWS) service pack [28]. AWS offers computational infrastructure resources in the form of virtual machines, but it also offers numerous managed services, that accompany mere cloud infrastructure. Amazon largely created the market for elastic cloud infrastructure with AWS. Before EC2 there was no such offering as one hour of computing power in the form of a Linux server [13]. In the heart of AWS is Amazon Elastic Compute Cloud (EC2). EC2 provides resizable computing capacity with server instances that can be obtained and started in few minutes. Scaling capacity up and down is easy and can be also automated.

AWS cannot be considered as pure IaaS cloud, because AWS includes so many managed service offerings. It is notable however, that most managed services are still separate offerings and charged separately. Amazon is not exceptional in this matter. Microsoft Azure cloud has been characterized as half-way between IaaS and PaaS cloud [6]. Amazon offering is however distinctively unique. Provided services in IaaS cloud do matter. Services are acquired in self-service manner and there is no customizing of services. On the other hand hosted services are so convenient and affordable, that there needs to be very good reason for not using them.

6.2 Regions and availability zones

Amazon offers three different regions in United States, one in South-America, two in Asia and one in EU. Data per se stays in region of choice. Pricing of services is also

done for each region separately. It is possible to use services and instances in several regions simultaneously using same AWS account, but data transfer between regions is charged. Each region has its own DNS endpoint for each service. For example EC2 has endpoint *ec2.us-east-1.amazonaws.com* in US East Region and endpoint *ec2.eu-west-1.amazonaws.com* in EU Region. Similarly Amazon Relational Database Service (RDS) has endpoint *rds.us-east-1.amazonaws.com* in US EAST Region and endpoint *rds.eu-west-1.amazonaws.com* in EU Region. Except for content distribution service CloudFront, which is a global service, basically all other Amazon cloud services are regional.

Each region is divided into three availability zones (AZ), which are used for designing high availability for cloud. From programmer point of view availability zones are indifferent. Every availability zone provides the same services. User doing deployment can choose availability zone or let EC2 to choose availability zone automatically. Availability zones are engineered to be isolated from failures in other availability zones. Although they are distinct locations within a region, they provide inexpensive, low latency network connectivity to other availability zones in the same region [29].

6.3 Amazon elastic compute cloud

6.3.1 Virtual instances

Launching virtual machines is based on images, read-only copies of the initial state of instances. EC2 started by offering Linux as operating system of machine images for instances. Currently Windows Server operating system is also available besides several flavors of Linux. Used machine image can be pre-configured template, Amazon Machine Image (AMI), to get up and running immediately or user can create own AMI containing applications, libraries, data, and associated configuration settings for example. User is also able to utilize his existing machine images by importing them to EC2 with VM Import/Export feature of EC2. This makes possible to utilize existing investments in virtual machines that meet set requirements for IT security.

Virtual machine by itself is not enough. Other things like block storages and public IP addresses typically need to be provisioned before instance is useful. All management in EC2 however, can be executed from command line. Because of this, provisioning can be done with a script. Possible ways to deliver a script are embedding it into machine image or passing it in the command that will launch the instance. Also, when instance is

launched by automated action, it can be passed a provisioning script at launch time by storing the script beforehand. Security and network access can also be configured on EC2 instance.

EC2 instances are of many different sizes. Regular instance types come in five different sizes, micro, small, medium, large and extra large instance. Besides regular instances there are high-memory instances for high throughput applications. High-memory instances are extra large instances or bigger. High-CPU instances on the other hand are available for sizes medium and extra-large. The only available type of high I/O instances is of size quadruple extra large. Also there are categories of cluster compute instance types for high performance compute and cluster GPU instance types for highly parallelized processing [29]. Both come with sizes quadruple extra large or bigger than that.

EC2 virtual machine instances can have either 32-bit or 64-bit platform. 32-bit platform however provides only limited scalability, because 32-bit platform is available only in sizes micro, small and medium. Any bigger instances are available only with 64-bit platform.

Amazon has defined EC2 Compute Unit (ECU) for measuring CPU capacity. One EC2 Compute Unit (ECU) provides the equivalent CPU capacity of a 1.0-1.2 GHz 2007 Opteron or 2007 Xeon processor. Medium instance for example has compute capacity of two EC2 Compute Units powered by one virtual core with two compute units. Complete specification of medium instance type is:

- 3.75 GB of memory
- 2 EC2 Compute Units (1 virtual core with 2 EC2 Compute Units each)
- 410 GB of local instance storage
- 32-bit or 64-bit platform

6.3.2 Programming AWS

Applications for AWS are possible to program with several programming languages. There are SDKs for Java, PHP, .NET and Ruby. Applications, that use AWS SDK for .NET, can be hosted in virtual instances that have Windows Server operating system. Other SDKs can be used in either available operating system, Linux or Windows Server. There are AWS SDKs for mobile development in Android platform and iOS platform.

Mobile applications are able to access AWS infrastructure services and hosted services like storage, messaging and database services in AWS. Same services can also be accessed without given SDKs with almost any client that can communicate over HTTP, because any AWS service can be accessed also through web service interface. Simple scratchpads that have been built with HTML and JavaScript for debugging purposes are good examples of this. Mobile SDKs, however, are better for rapid development of mobile applications that for example upload photos and videos into cloud infrastructure.

6.3.3 Pricing

Basic pricing model of EC2 is On-Demand Instances that let customer pay for compute capacity strictly by the hour with no long-term commitments. There are optional pricing models that allow customers to optimize their expenses based on their expected usage. Besides virtual machine instances EC2 contains services, that are part of EC2, but they are still charged separately by the use. Those are Internet data transfer, Amazon Elastic Block Store, public internet addresses, Amazon CloudWatch and Elastic Load Balancing.

Internet data transfer is charged by amount of traffic going out of region. Other Amazon Web Services like Simple Storage Service (S3) for example are regional services. There is no charge for data transfer between EC2 and other Amazon Web Services provided that they are in same region. Data transfer between EC2 instances in different availability zones is not charged.

6.3.4 EC2 features

Public internet addresses in EC2 are called Elastic IP Addresses. Elasticity refers to IP addresses outliving any virtual instances. Instances might be obtained, terminated or restarted, but Elastic IP addresses preserve once they have been reserved in EC2 and can be attached to new instances. EC2 instances have also private IP addresses that can be used only inside EC2 region, but those IP addresses do not outlive instances.

Elastic Block Store is block level data store that provides characteristics like latency and reliability similar to Storage Area Network (SAN) [15]. Elastic Block Store is one of the several storage services in Amazon Web Services. Unlike Simple Storage Service (S3) however, EBS needs to be considered as inseparable part of EC2, because EBS is not transferable outside of EC2 region and it cannot be directly accessed outside of EC2.

Amazon Elastic Block Store outlives instances and can be attached to new or restarted instance, but simultaneously to only one instance.

Elastic Load Balancing (ELB) is an important feature of EC2. ELB distributes incoming application traffic across multiple EC2 instances. ELB provides significant fault-tolerance by detecting unhealthy instances within a pool and automatically rerouting traffic to healthy instances until the unhealthy instances have been restored [30]. ELB can be used in connection to auto scaling feature of Amazon CloudWatch. Fault tolerance in Elastic Load Balancing requires health check to detect unhealthy instances. While setting up an ELB, thresholds are configured to decide if an instance is healthy or not.

Amazon CloudWatch is monitoring for AWS cloud resources and the applications customers run on AWS. Basic level of CloudWatch comes with EC2 instances. Only detailed monitoring, that has service level exceeding basic level, is charged. Basic monitoring and detailed monitoring differ in frequency of measurements. Basic monitoring contains ten metrics for EC2 instances, eight metrics for EBS volumes and ten metrics for Elastic Load Balancer. Other Amazon Web Services like databases also include CloudWatch metrics. One free metric is also metric of estimated charges on user's AWS bill. The number of these metrics also depends on used services. User is able to create custom metrics using CloudWatch dashboard in AWS management console or with CloudWatch API and complement pre-selected metrics with them. Also user can set alarms to CloudWatch with specified thresholds. Alarms send notifications to user and are able to take other automatic actions. Resulting data from metrics can be viewed as charts or statistics that are available for user in AWS management console, but statistics data can also be fetched from EC2 with queries using CloudWatch API.

Auto scaling is feature in EC2, that lets user take more benefit out of cloud. Auto scaling takes at the same time most of the work out of users' hands. First benefit from auto scaling is simply scaling. Scaling capacity automatically up and down makes the most benefit out of elasticity of cloud. Scaling up and down requires measurements in order to enable triggering automatic actions. CloudWatch provides those necessary measurements. User is able to specify rules for automatic actions based on metrics. Trigger, that user defines for action might be for example certain level of CPU usage. Launching and terminating instances automatically requires an image that launches into

an instance that can independently share load of application. Typically load is divided by load balancer. Unit, that auto scaling operates with is auto scaling group. Auto scaling group can be set up to add and remove these instances from the load balancer. Auto scaling group requires launch configuration, which is prerequisite but also an important tool. A launch configuration determines what kind of instance is being launched. It sets the AMI and the type of instance that is going to be created. Launch configuration may prescribe arbitrary number of availability zones inside a region. The auto scaling group balances the instances evenly over the availability zones prescribed in launch configuration. If auto scaling group is hooked up with an ELB, instances will always be removed from the load balancer before being terminated. This is to ensure that the service level of the load balancer in not degraded. The power of availability zones becomes apparent, if happens that request to start a new instance cannot be met, for example due to capacity problems. If auto scaling group is configured into multiple availability zones, it will launch an instance in another availability zone. Auto scaling group will try rebalancing of instances between availability zones later [13]. Another benefit of auto scaling is, that it makes application more resilient. Auto scaling can be used for monitoring only one instance for example without using dynamic launching. If the instance dies, however, auto scaling can launch a new instance automatically. This same technique can be applied to multiple instances distributed over different availability zones.

6.3 Hosted services

6.3.1 SQS

Amazon Simple Queue Service (Amazon SQS) is a distributed queue system that enables applications to queue messages that one component in the application generates to be consumed by another component. This is a means of decoupling application. SQS move data between distributed components that perform different tasks, without losing messages or requiring each component to be always available.

Atomic locks are used in reading messages in order to keep multiple readers from processing the same message. It is possible, that many writers are writing to a queue at the same time. SQS does preserve the order of message in queue, but the distributed nature of SQS makes it impossible to guarantee it. This is a trade-off in a massively scalable service, what SQS is [13].

SQS, like most hosted services in AWS, include web service API. API consists of simple HTTP requests containing actions and their parameters URL encoded. This simple web service API that Amazon calls "query API", is only web service API for SQS, since SOAP interface, which is provided for many AWS services, is not provided for SQS. Amazon SQS Scratchpad is easy HTML and JavaScript based debugging tool that uses SQS API. SQS Scratchpad is noticeable, because SQS is one of the few services in AWS that do not have command line tools provided by Amazon. Figure 8 shows, how new queue was created with SQS Scratchpad and how SQS Scratchpad displays the URL, that was sent over HTTPS and created the new queue. Access key id and secret access key have been erased from the figure.

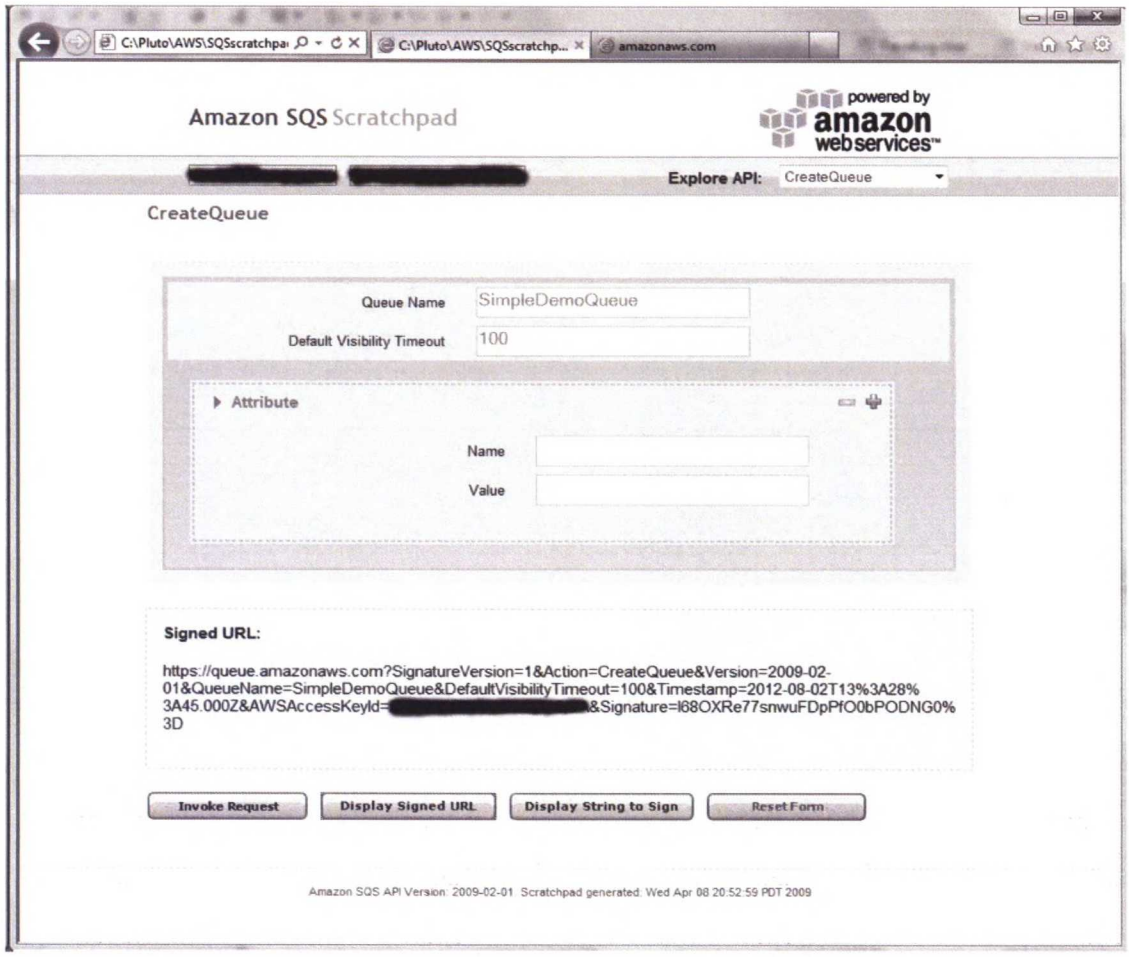

Figure 8, scratchpad displays URL

Domain name in encoded URL is queue.amazonaws.com, which is the endpoint of SQS service in Virginia, USA. Queue created in US-east region is not visible in US-north-west region or EU region. The last parameter in encoded URL is signature, which has been cryptographically created with secret access key. Signature ensures, that only user, who possesses secret access key, can access resources with AWS account. Reply from

API call acknowledges successful operation. Figure 9 shows XML, that API returned from HTTPS get request.

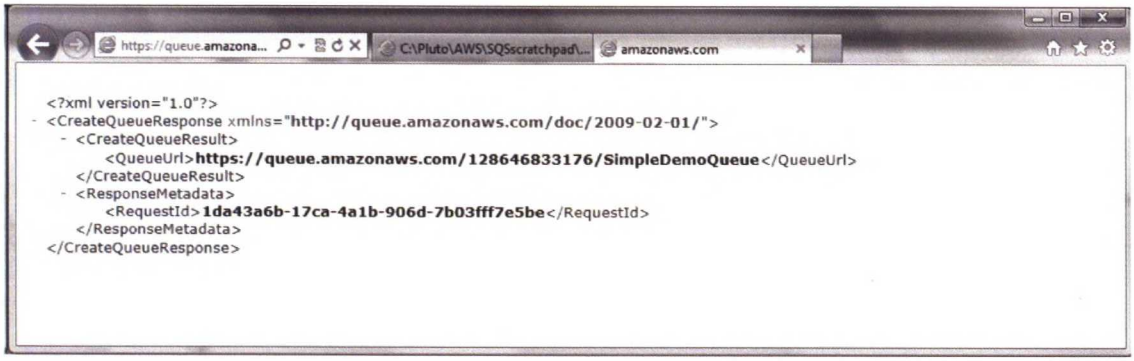

Figure 9, response from API call

Programming EC2 can utilize different tools and programming languages. Eclipse Integrated Development Environment (IDE) is open-source tool for software development. Many, if not most, software developers using Eclipse, use Java programming language in development. Java is only one of several programming languages that Amazon Software Development Kit (SDK) accompanies. One convenience of Eclipse is AWS Toolkit for Eclipse that functions as an Eclipse plug-in and eases programming EC2. Figure 10 shows setting access key into AWS Toolkit for Eclipse. Access key is necessary for remote use of Amazon Web Services.

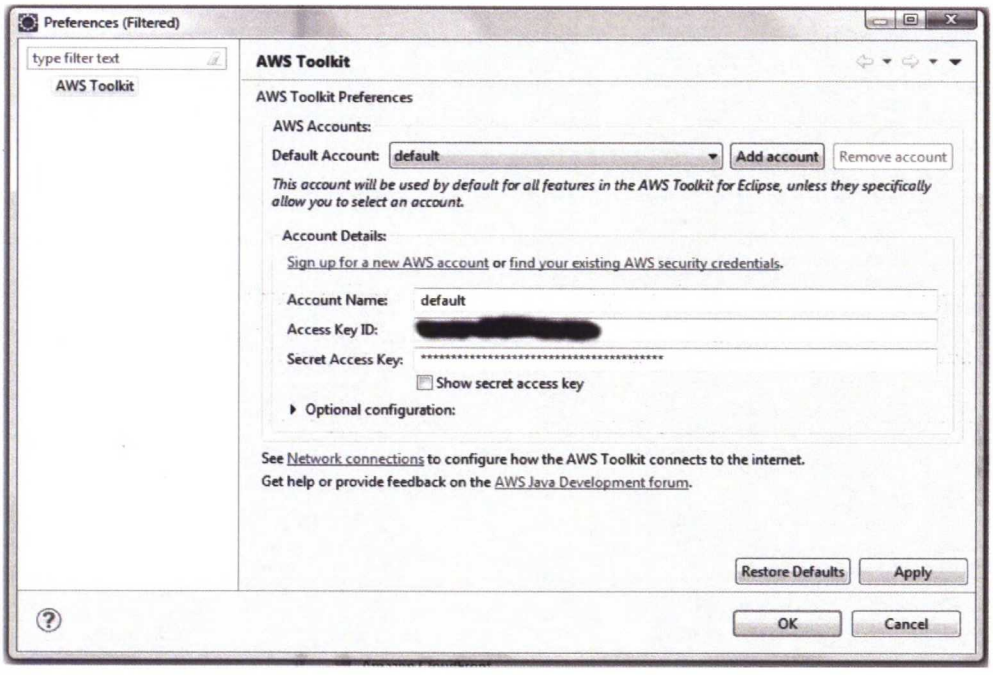

Figure 10, AWS Toolkit preferences

AWS toolkit enables browsing resources in AWS remotely directly from development environment. Figure 11 shows, how AWS browsing in Eclipse shows newly created SQS queue after refreshing display.

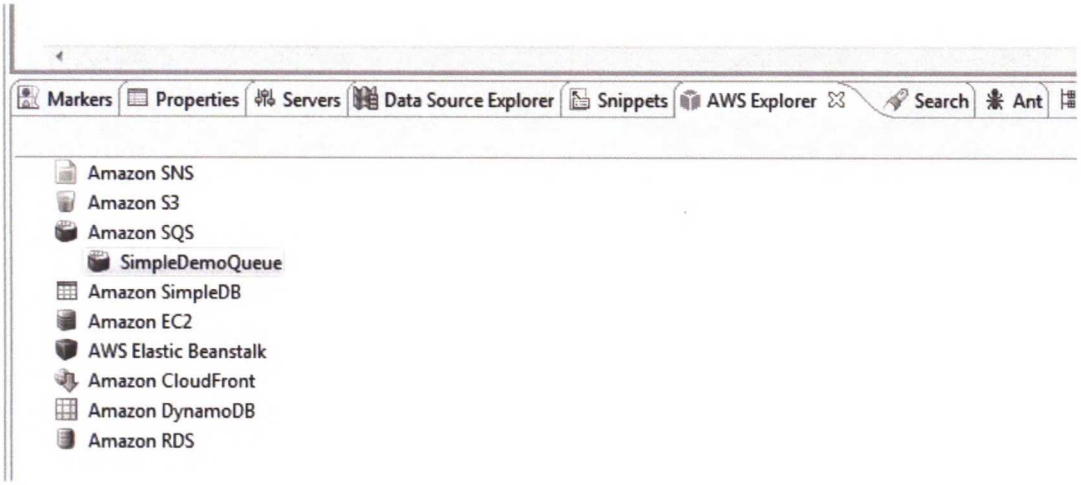

Figure 11, browsing AWS Explorer

6.3.2 Simple Storage Service

Amazon Simple Storage Service (S3) is storage service for large and file like data. S3 is scalable up to web-scale. Objects in S3 can be made private or public, and rights can be granted to specific users. Particularly important feature of S3 is, that objects are directly HTTP addressable. S3 provides a simple web services interface, that can be used to store and retrieve any amount of data from anywhere on the web. Although objects are addressable from anywhere, each endpoint belongs to particular region. For example objects stored in the EU Region never leave the EU, unless owner of objects transfers them out.

S3 has service lever agreement (SLA). According to agreement AWS will use commercially reasonable efforts to make Amazon S3 available at least 99.9% during monthly billing cycle. Three nines availability means at most 43 minutes outage during one month. Amazon S3 uptime is however calculated by using 'error rate' and 'monthly uptime percentage' that are defined by AWS [31]. 'Error rate' is defined as percentage of failed service calls, returned by Amazon S3 as error status "InternalError" or "ServiceUnavailable", calculated during period of five minutes. 'Monthly uptime percentage' is defined as subtracting from 100% the average of 'error rate' of every five

minute period during one month. This can be clarified with two examples. Supposing in first example, that particular user hits the service 100 times per second and 1 in 200 service calls fails, uptime would be 99.5%. That is below committed level. Second example is, that user hits the service 10 times per second and every call succeeds, except during 30 minute period, during which user hits the service 1000 times per second and only 10% succeeds. Uptime in second example would be 99.94%, which is above committed level. Although SLA has these measures, Amazon informs, that S3 has been designed for 99.99% availability.

This far S3 has seen an exploding growth.

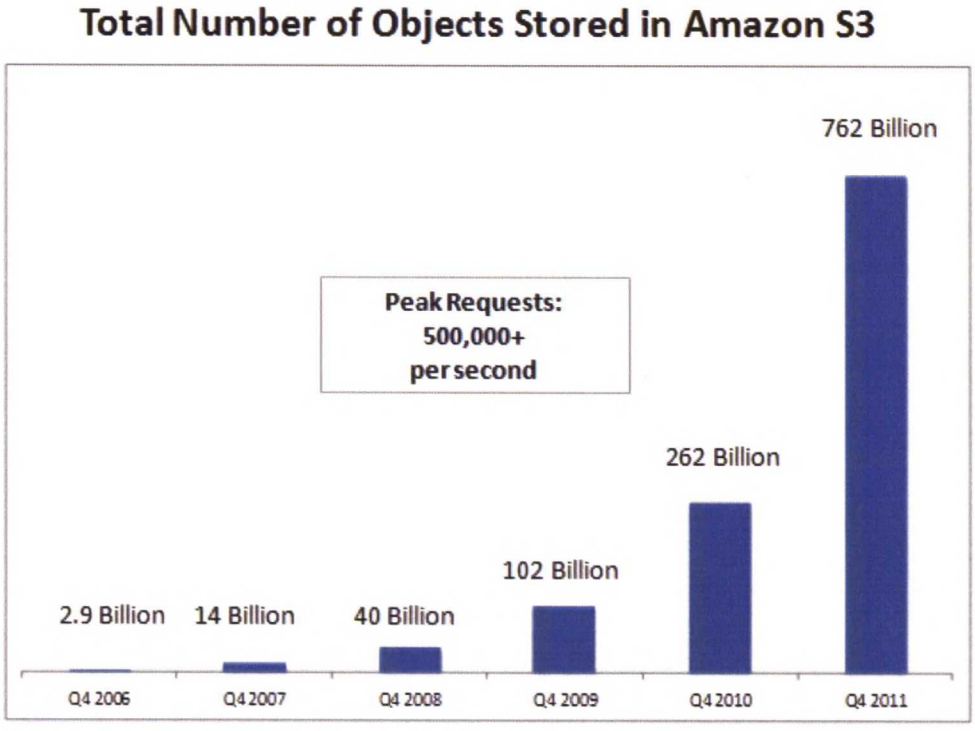

Figure 12, growth of Simple Storage Service
Source: Amazon Web Services Blog, 31.1.2012. [1]

6.3.3 Relational Database Service

Amazon Relational Database Service (RDS) provides relational database as a hosted service. RDS is a familiar relational database in every aspect and introduces few new requirements for its user in order to make database run in cloud. User has three different database engines to choose from, MySQL, Oracle and Microsoft SQL Server. RDS is a

[1] *Available: http://aws.typepad.com/aws/2012/01/amazon-s3-growth-for-2011-now-762-billion-objects.html*

scalable service and database instances have instance classes similar to virtual machines in EC2. RDS instances start from micro DB instance class and go up to quadruple extra large DB instance with high-memory and high I/O capacity. RDS provides exceptional relational database, because it can be moved into bigger DB instance class while already using DB instance. This causes brief outage, which can be scheduled to maintenance window.

DB instance of RDS is a managed database service and not dedicated machine and it does not allow direct host access via Telnet, Secure Shell (SSH) or Windows Remote Desktop Connection. What RDS does support however, is access from any standard SQL client application. Example of such application is *mysql*, client program of MySQL. Since there is direct access to a native database engine, tool designed for particular engine works with Amazon RDS. Amazon provides command line tools that can be used for managing RDS instance. Other possible interfaces for RDS management are HTTP query API or SOAP web services messaging protocol.

RDS is powerful, because it largely frees user from the burden of optimization, backups and scaling. It is a service that can provide high availability and high durability. RDS handles backups and recovery of data if not disabled from doing that. Basic recovery mechanism of RDS is automatic host replacement, which is available for basic deployment also, if backups have not been disabled by user. Availability of RDS is optionally strengthened with multi-availability zone deployment.

Backups in RDS are basically automatic and managed by RDS. User has the possibility to restore the state of database instance to any moment in time with point-in-time recovery. Going back in time is made possible by backups and the possible timespan is limited by backup retention timespan. DB snapshots are user-initiated full database backups, but they are independent from backups, that RDS creates automatically following backup preferences of DB instance. Unlike automatic backups, snapshots are stored indefinitely until user deletes them. They can be stored also after database instance itself has been terminated.

RDS enables multi-availability zone deployments, which are currently supported for the MySQL and Oracle database engines. Multi-availability zone deployment creates redundancy into executing database instance. Created second replicated database instance is located in other availability zone for higher availability. It is notable, that

only one replica is still used for both write and read. Standby replica is not available for either operation. Availability zones do not have dependency and also failures in different availability zones are expected to be not dependent. This is the benefit of deploying database instance into different availability zones. RDS is a service that manages fault-tolerance in multi-az deployment and executes automatic failover in case of emergency and during maintenance windows. Multi-az increases availability of database also during planned operation and not only during emergency. Maintenance and both the automatic database backups and manual snapshots cause some outage to RDS. RDS with multi-az deployment conducts backups and maintenance on the replicated DB instance. After maintenance has been conducted on the standby replica, RDS automatically performs a failover to do the maintenance on the other instance. Except for short period of time during failover, database service does not experience outage, because standby replica is a hot-standby processing node. In normal situation there is no wait until standby replica catches the primary instance, because standby replica is not behind. RDS treats both database instances symmetrically in failover protocol. When backup replica becomes master database instance, the other instance becomes backup replica. Replication in multi-az deployment is synchronous. It executes in context of a transaction. This means also that multi-az deployment does not lose data during failover. This is true even when failover is not planned.

Read replica is another type of replication, which is available only for MySQL database engine in RDS. Read replica however has completely different purpose. Replica is created for read scaling. Read replica allows several database replicas to distribute and balance load of read operations, but does not allow write operations to other replicas except primary replica. This replication protocol is asynchronous and it induces significant replication lag. Because of this, read replica does not provide any improvement to data durability and it does not provide fault-tolerance mechanisms like multi-az deployment does, although read replicas can be created also with multi-az deployment. Basically read replicas introduce eventual consistency in database, because of lag in replication. This does not mean however, that there would be write conflicts and weakened durability like in NoSQL database, because write operations can be done only to master replica. Read replicas just introduce the old type of eventual consistency that has always been there. In fact read replicas in RDS are implemented with MySQL's native, asynchronous replication.

6.3.4 SimpleDB

Amazon implemented as an internal system non-relational database called Dynamo. Later, Amazon SimpleDB that builds on Dynamo was launched as an AWS product. Latest development is that Amazon has publicly launched database service that is also called Dynamo. SimpleDB is a non-relational database, because tables in it are isolated entities without the possibility of doing join operation in database. Joining data in different tables, or domains, as they are called in SimpleDB, is possible in application by doing several round trips into database. Because join operation is not possible in database, complex queries are not possible. This has inspired to call SimpleDB and any similar non-relational database as NoSQL. Data in domains, SimpleDB tables, is in a structured form but does not adhere to schema. Instead of columns, SimpleDB domains contain attributes, equivalent of columns in relational database table. Another consequence of non-relational nature of database is that complex transactions are not supported by database. SimpleDB is a managed service and it has the advantage of being ready to use right away. There cannot be much simpler provisioning and setup than that.

Idea behind NoSQL is mostly scalability. After giving up the possibility to execute join operations in database, scaling up becomes easier and in fact does not require any further measures [13]. Scalability makes NoSQL very fit to cloud. Scalability does not however come without price. As it has been proven, that consistent, available and scalable (partition tolerant) shared data is not feasible, design is necessarily a trade-off. In the case of SimpleDB, notable is eventual consistency. SimpleDB reserves the right to take some time to process the put/delete operations. This is not about latency or throughput, but about differences in data, that is concurrently read from different sites of distributed database at some particular point of time. After put operation returns, old value of data might be read from some other distributed part before data propagates into all sites. Eventually data will end up in consistent state, but there might be significant lag and data durability might be compromised in case of write conflict. SimpleDB does provide also consistent read and conditional put/delete to get around this, but those operations are more expensive than basic operations.

Dynamo seems almost identical to SimpleDB. Dynamo builds indexes only for primary key attribute or attributes whereas SimpleDB creates indexes automatically for every attribute. Fundamental difference in Dynamo compared to SimpleDB is, that Dynamo

has been designed for optimized performance. Both database services are charged by use like always, but pricing is different. Dynamo has been designed for high and predictable performance and its use is charged by provisioned throughput capacity and by storage capacity, which is more expensive than in SimpleDB. Dynamo does not have restrictions for data size like SimpleDB does.

Chapter 7

Marketing software system

7.1 Overview

This study examines one particular software system, which has been deployed to cloud by case company. System is a mobile marketing application. It is administered by the case company, which is a service company that offers mobile marketing service to retail companies. Retail companies provide marketing offers that are published to users. Consumer users on the other hand subscribe themselves to retailers and topics, which they are interested in. They receive messages to their mobile phones and use web portal to set their personal data and configuration. Consumer application has thus two different clients, mobile client and web client. Mobile client has also several versions that are available for different smart phones. Major smart phone operating systems are covered with mobile applications that execute native code in mobile operating system. Other mobile platforms are covered with Java Micro Edition based client program. Mobile applications are available for download in mobile platform specific application markets.

Besides consumer application system includes content management application for updating system content and management of system. Content management application has web user interface.

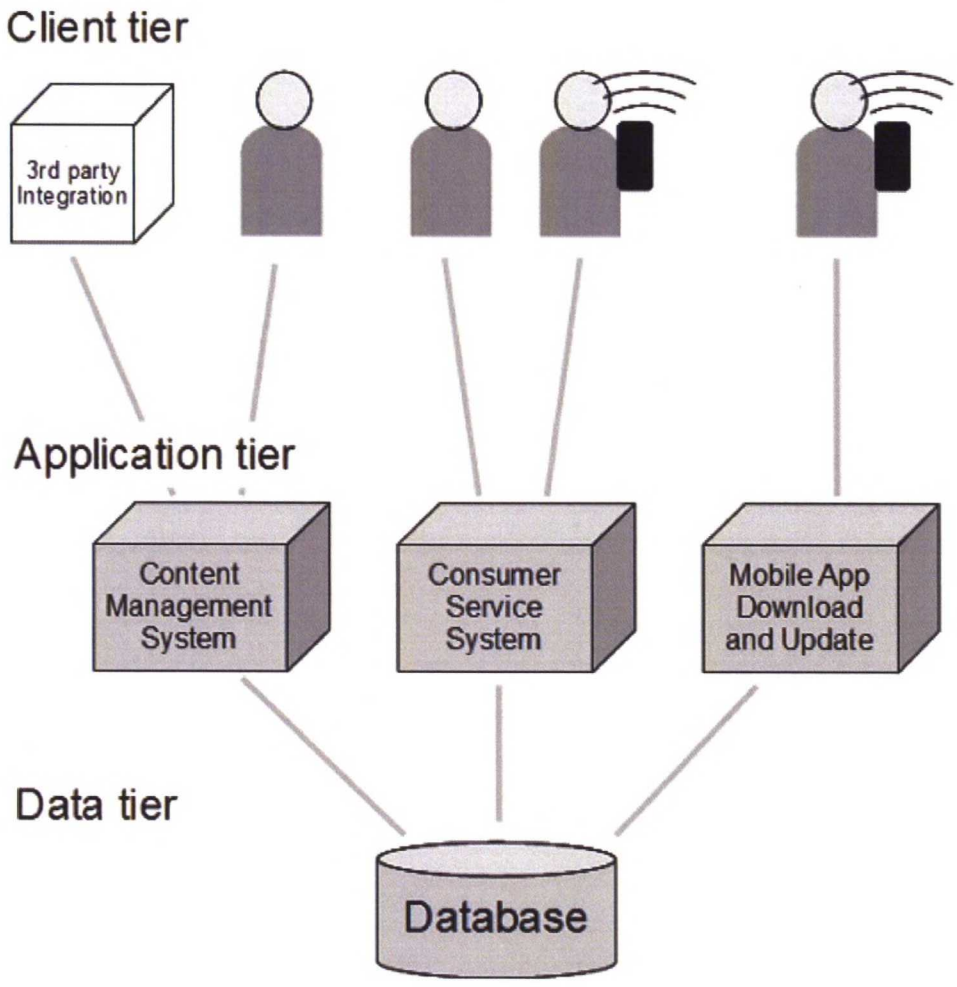

Figure 13, overview of deployment

Figure 13 has been adapted from architecture documentation. It shows all separately executing sub-systems. Application tier of mobile marketing application has three separate sub-systems. User of consumer service system has two different user interfaces for the same application, web client and mobile client, which is a fat client. All server modules are deployed into cloud infrastructure. This means, that every client that system has, is actually a cloud client. All cloud clients connect with HTTP into cloud, also fat client in mobile device, which is not hypermedia driven client. Server side thus serves web traffic. Also third-party integration is implemented with web service interfaces, only with the difference that it uses SOAP protocol, which is built on top of HTTP.

7.2 Consumer application architecture

Module, which serves consumer customers, is special, because it has multichannel access. Mobility is what creates multichannel access and mobility is known to be one

driver for cloud computing. Cloud computing for larger public means exactly this kind of applications. Consumer application has mobile clients that are installed applications. Some versions of mobile clients execute native code and some cross-platform code. All of them are anyway installed client programs. Such client programs are also called fat clients to distinguish them from so called thin clients, which are user interfaces in web browser and executing only HTML and optionally some program code that is downloaded on-demand to web browser. Application tier of the consumer application on the other hand, contains only one server application. Server side even has interfaces that are mostly shared by different types of clients. Sharing interfaces between different types of clients is convenient, but it requires planning.

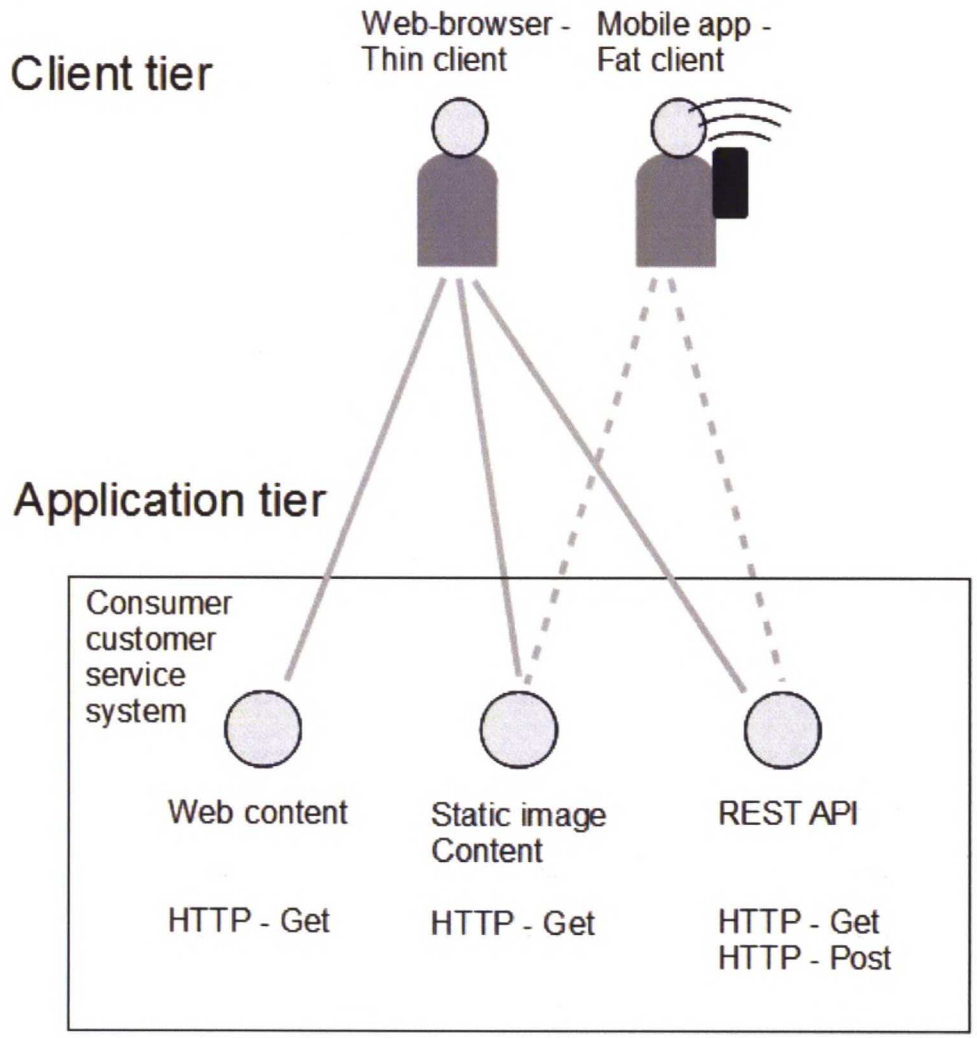

Figure 14, Client interface in consumer service sub-system

Figure 14 has been adapted from architecture documentation. It shows below interfaces all HTTP methods that are used in each interface. Caption REST API in figure 14

denotes interface, which is used for manipulating resources in system. In other words it is used for adding, deleting and updating data that is represented in clients. In principal each of the three interfaces are REST-compliant and not only this one. Caption Web content denotes interface, which provides dynamic web content. Provided web content is representation of the resources in the system, but interface does not manipulate the resources. Caption Static image content denotes interface, which provides only images.

Clients use Ajax requests to communicate with REST API. Ajax request is just like any other HTTP request except for its origin in client. Ordinary HTTP request is initiated by user action and through HTML representation. Hypertext-driven HTTP request is represented as a link or form submission in hypertext. Link in hypertext denotes user initiated transfer to other page and to other state. Ajax request does not have this kind of semantics. Instead Ajax request is initiated by executable code in client. Executed code is arbitrary and event that triggers the request is also arbitrary. State transitions are nevertheless initiated by client. Web browsers have the convenience of executing Ajax requests asynchronously. Term Ajax denotes usually more broadly techniques of JavaScript programming that use asynchronous HTTP requests to build web applications. Similar HTTP requests can be done by a client, which is some other program than web browser and programmed with Java or some other language. This is the case with the system that serves consumer customers. It has clients, namely in mobile platforms, which do similar HTTP requests as web browsers do, although those clients are not web browsers. This is a clever way to create a uniform interface between clients and server. Ajax requests are special, because the content that server sends in reply to HTTP request is not hypermedia content but mere data instead. This is natural, because Ajax call does not have the semantics of page transition in web browser. Client updates user interface accordingly, when receiving data from server as a reply to Ajax request. When client is web browser, this means updating hypermedia content in browser dynamically with JavaScript code. When client is mobile client, it means updating user interface in client program.

Figure 14 shows interface, where web browser requests web content, HTML and JavaScript code that functions as a graphical user interface of the application. Figure 14 shows also that mobile clients do not request web content. This is because they are not hypertext-driven. This interface is read-only. All updates into server go necessarily through REST API in order to preserve uniform interface between client and server.

Fat clients in mobile platforms have the drawback that all updates into client programs needs to be explicitly installed into mobile devices. Mobile marketing application has separate sub-system for serving mobile client installations and updates.

7.3 Cloud deployment

7.3.1 Infrastructure

The whole system has been deployed into IaaS cloud, that Amazon offers, Amazon Web Services (AWS). Deployment uses infrastructure services that provide computing power, network bandwidth and storage capacity. Infrastructure services are provided by Amazon Elastic Computing Cloud. Besides infrastructure, deployment uses also Elastic Load Balancing (ELB), Simple Storage Service (S3) and Relational Database Service (RDS), that are some of the fully managed services that AWS provides. The whole deployment is in Amazon's region 'EU', which resides in data centers in Ireland.

Application tier consists of Linux servers that have Tomcat and Oracle's Java runtime installed. Apache Tomcat is a cross-platform web server, which contain servlet container. Data tier on the other hand, is solely deployed to RDS. RDS does not contain any user manageable instances. Only application specific data is deployed to RDS. User of RDS sees database instance in RDS similarly as any database engine, although not in his own total control.

7.3.2 Processing nodes

Deployment has some amount of redundancy of processing nodes. Each application tier server is duplicated and connected to ELB. Both server instances are actively processing client requests and ELB divides the load between them. Redundancy of server nodes utilizes availability zones in EC2, so that each virtual machine instance is started in different availability zone than its pair redundant machine. Region has three availability zones, but there are only two redundant servers, which so occupy two of the total three availability zones. RDS has also redundancy that is somewhat similar to redundancy of web and application servers deployed into virtual machines. RDS is used in multi-az mode, which has two replicas of database in two sites in different availability zones. While RDS contains redundancy, it has important difference compared to application servers. The difference is that all data updates are copied into both database replicas, whereas application servers have been designed to be stateless. They don't have any data

that is not preserved in restart. Application servers are thus independent of each other.

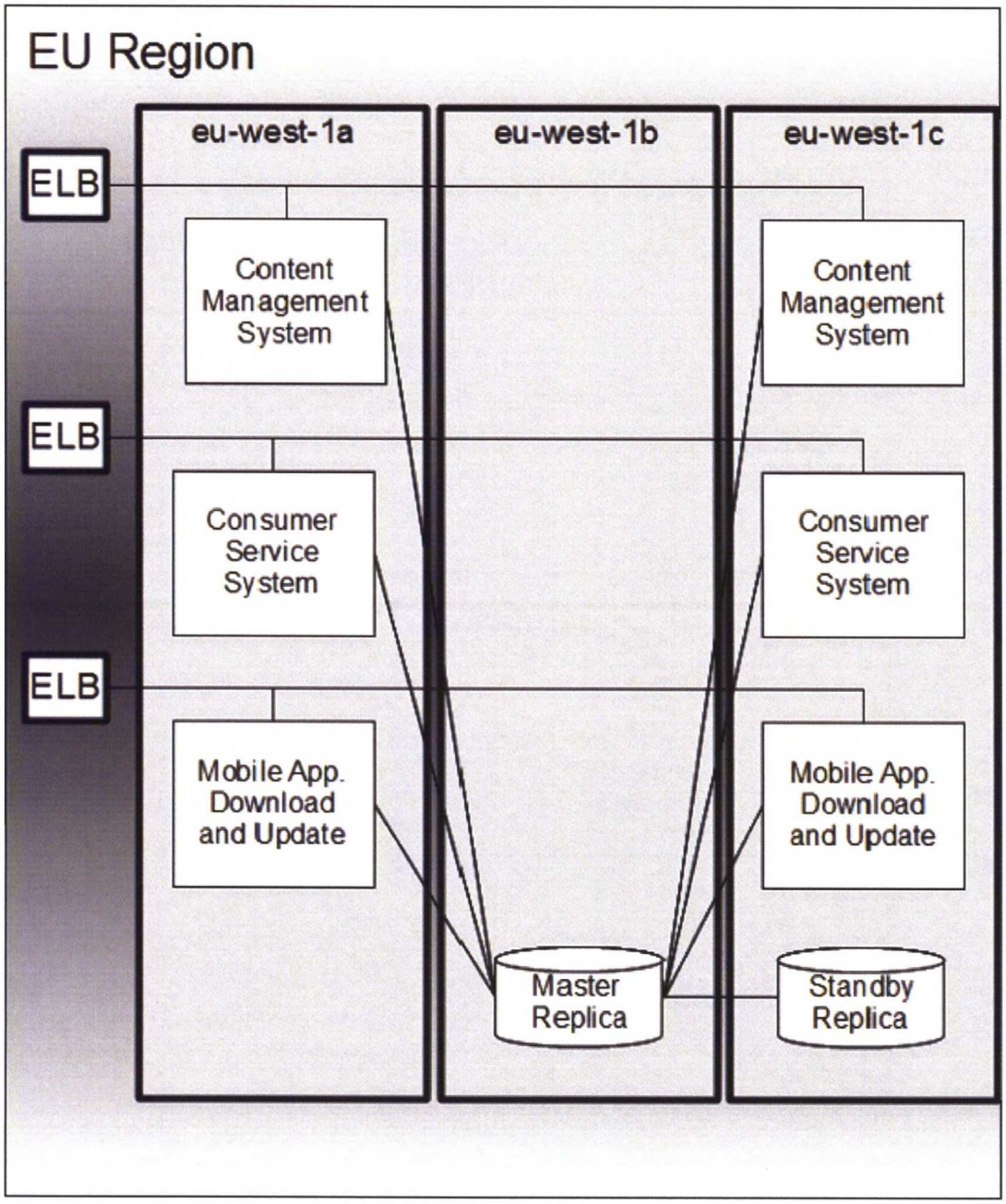

Figure 15, redundancy in cloud deployment

All redundant nodes in deployment are shown in Figure 15. Region contains three availability zones and each node is configured in start-up to start in certain availability zone. Regions in AWS have low latency network connection between availability zones in same region. Data transfer between zones is also not charged. ELB is shown in figure 15 three times in order to show, that each server class is referred with different domain name.

7.4 Failures and recovery

7.4.1 Detecting failure

User of cloud does not have control over infrastructure and direct observations are not possible. Nevertheless dependability requires that unmasked failures of any used resources are detected in a way or another.

Amazon EC2 performs automated checks on running instances to identify hardware and software issues. There are two types of status checks: system status checks and instance status checks [32]. These status checks can be used for monitoring instances in a similar way than CloudWatch metrics are used. Status checks produce results that indicate either healthy or impaired instances. At the same time they possibly give information about maintenance event that may be scheduled by AWS. Status checks can be viewed from management console or with command line tools. Also information can be fetched with web service API. Similarly as with CloudWatch, alarms can be set to status check events.

Both checks, system status checks and instance status checks, have same kind of purpose, but they check failures, which have different causes. The cause of failure determines how the problem can be repaired. System status check detects problems in AWS systems, which are needed for using virtual machine instances in AWS. Instance status check, which is the other type of automatic check, detects problems in user's particular virtual machine instance. These problems reside in software and configuration since instances are virtual machines. Examples of such problems are exhausted memory and start-up misconfiguration. Distinction between individual instance and underlying AWS systems is important consideration. Infrastructure is in cloud and it is outside the user's control. Examples of problems in underlying AWS systems, that system status check detects, are loss of network connectivity and hardware issues on the physical host. In case that system check status is impaired and problem has been detected in AWS systems, repairing the problem requires AWS involvement. Amazon however suggests anyway, that user may also try to resolve the problem himself by restarting or replacing the impaired instance depending on instance type.

Failure mode is equal to output of the system in case of service failure. Certain failure mode of virtual machine instances is fail-silent mode, where no connection to virtual machine can be established. In case, where either system status or instance status is

impaired, failure mode can be expected to be fail-silent. Software in instance would not work properly and no reply from instance would be received. If this failure mode is the only occurring failure mode, system is fail-controlled. Failure of EC2 instance could however also be non-silent. Example of such problem is hardware issues on the physical disk underlying EBS volume. Corruption of file system in virtual instance is an example of problem, which would make instance status check to fail and also silence the instance. Single disk error in EBS volume on the other hand would not necessarily cause either effect.

Case company monitors marketing application in production environment with two different monitors, Amazon CloudWatch and Zenoss cloud monitor. Amazon CloudWatch is service provided by Amazon and part of Amazon Elastic Computing Cloud. This is why CloudWatch provides different information than Zenoss, which is outside of cloud and resides in own premises of case company. Both systems monitor health of virtual instances. Zenoss measures response times from the nodes through the external interfaces of EC2 and detects unresponsive nodes this way. CloudWatch on the other hand reports about the health of the virtual instances from the point of view of ELB load balancer. This has at least the benefit of providing information that is consistent with the internal state of ELB, which contains fault-tolerance mechanisms. CloudWatch also monitors resource usage of virtual machines, namely processing load, disk usage and memory consumption. Zenoss measures performance and availability of processing nodes through the external interfaces of EC2 from response times. Both give notices about failures to administrator, Zenoss with SMS and CloudWatch with email.

7.4.2 Recovery

Failure of virtual machine instances is event of failure that is expected to happen. Designing recovery from failure must be based on anticipation of failure mode. In case of Amazon EC2 instances, silently failing instance can produce two different outcomes: instance state becomes terminated or either system status check or instance status check indicates failure. In case that failure is caused by AWS system, which would also be indicated by impaired system status, AWS involvement might repair the problem. If repaired problem was network failure and instance was running but could not be connected, there would be possibility of full recovery of instance without restart of instance. It is possible however, to treat network partition and instance failure like they

were the same thing and examined system does exactly that. Status of an instance is treated as binary in the system. It is either running or failed. Several reasons favouring this approach can be found. First, while problem is occurring, network partition and instance failure are indistinguishable, if instance failure is silent. Second, network fault might cause some damage to system. Restarting the instance returns the node to the initial reset state, which is a predetermined state. Returning node into a known state by restarting is backward error recovery and repairs possible damage, that network fault might have caused. Conclusion is, that it is good to have only one kind of recovery, which is restarting or replacing, depending on instance type. Restarting an instance in cloud is different than restarting physical appliance. Cloud relies on virtualization for management and is built to be managed as a whole. So it is possible to shift execution into a functioning part from a non-functioning part. When shift is successful, restarting is able to repair for example loss of system power, which is a physical problem.

Case company implemented script that automates tasks of recovering failed instance. This implementation, however, needs administrator involvement. Monitoring sends SMS about failed instance to administrator, who initiates instance recovery.

7.4.3 Fault-tolerance in software services

Case system contains such parts as ELB and RDS that are services managed by cloud service provider. Both of them have fault-tolerant behaviour built in them. RDS provides fault-tolerant database instances with notable ease.

ELB has built in functionality for detecting unhealthy instances within a pool and automatically rerouting traffic to healthy instances. This ensures continuation of service, when some instance is not working properly. ELB uses watchdog timer to implement health check. Each server needs to reply health check message, which ELB sends autonomously. Health check detects lost or locked out components, but it has potential also for more subtle detection of failure. It is up to application to implement logic, which replies to health check message. Checks, which application runs, create possibility to detect other than fail-silent failures when instance is lost.

ELB is an intermediary between web client and server, which makes possible to dynamically substitute server. This is made easier by the automatically rerouting of traffic to healthy instances. Dynamically substituting server is possible also while using HTTPS, because ELB can establish SSL session and thus become SSL endpoint.

HTTPS uses cryptography to encrypt messages, which makes impossible to take over SSL session. Rerouting of traffic to healthy instances would be impossible, if server was SSL endpoint instead of ELB.

Fault-tolerance in web application is possible because multiple servers can provide same service and a single server can be dynamically substituted. REST compliance in architecture is key factor in this. REST makes possible processing by intermediaries. It makes also possible creating several redundant copies of servers. Statelessness is also one of the constraints of REST and it makes failover possible. Load balancer is central element in architecture of the mobile marketing application. It acts as an intermediary that supervises servers and performs dynamic failover.

The mobile marketing application has its database hosted in RDS, which is managed service. Used multi-az deployment mode has redundant database replica as backup-replica. RDS performs failover of database sites in case there is a failure of database master-replica. Failover is implemented by updating DNS to point to site, that hosts database backup-replica. Database connections in application are replaced with connections to new site that hosts master-replica.

Chapter 8

Cloud outages

8.1 Estimating cloud outages

At least the biggest of public IaaS cloud providers give guarantees to their service level and availability. Amazon, which is market leader in IaaS cloud, provides service level agreement for its Elastic Computing Cloud offering that promises an annual uptime percentage of at least 99.95% during a service year [33]. Amazon defines uptime percentage in a rather complicated way. First, downtime means that region is unavailable and because region consists several availability zones it means that more than one availability zone in region is unavailable. Availability zone is unavailable when virtual instances cannot be connected and new instances cannot be started as replacements. Final availability percentage value is calculated from the proportion of five minutes periods when region is unavailable. Windows Azure is comparable to AWS and it also has SLA [34]. Microsoft declares SLA in a similar manner than Amazon does and with equal number, which is 99.95%. Microsoft's SLA differs from Amazon's corresponding SLA when it refers to recovery mechanism that is built in Azure and Microsoft's direct involvement as cloud provider in recovery of instances. This is difference compared to Amazon, because recovery of instances relies on user's involvement in EC2, when no additional services are used. Guarantees are given to deployment of at least two server instances in different fault and upgrade domains. Considering that Amazon's EC2 regions are divided into availability zones and that Amazon's guarantee applies to two out of three availability zones, Microsoft's SLA is very similar to Amazon's. Microsoft also guarantees each service with separate agreement, which is similarity to Amazon. Guarantee of 99.95% service level is given to

server instances. Other services have guarantee of 99.9% service level. Curiously French company OVH gives astonishing 99.99% SLA for its public cloud [35].

The International Working Group on Cloud Computing Resiliency (IWGCR) has a mission to monitor and analyze Cloud Computing resiliency. This Paris based organization was formed on March 23, 2012 by Telecom ParisTech and Paris 13 University and is composed of IT executives, academic researchers and industry representatives. IWGCR [36] published on June 18, 2012 the first Availability Ranking of World Cloud Computing (ARWC). ARWC report aggregates information about availability of cloud computing services during years 2007 – 2012. It provides an initial estimate of the average availability of cloud computing services. This short report is remarkable because it is the first of its kind. Results show downtime and availability of 13 service providers. Results show an average of 7.5 hours of downtime of cloud service per year, which equals approximately 99.9% availability. It is noticeable, that of all 13 evaluated could service providers only four have estimate lower than 99.9%. Providers of IaaS cloud services are virtually impossible to compare based on report. Amazon is the only significant infrastructure service provider included in ranking. Microsoft for example offers variety of different services. Windows Azure, which is Microsoft's infrastructure service offering, started as recently as 2010 and its share in Microsoft's services has not been specified in any way in ARWC report. Curiously Amazon and Microsoft take 8^{th} and 9^{th} place in ranking respectively. Report shows average availability of 99,954% for Amazon and 99,941% for Microsoft.

Report has been aggregated from information that has been collected from press releases. It is admitted in report, that used means for gathering information are not exhaustive. Gathering press releases does not achieve good coverage. Another shortcoming is that average availability calculation does not take into account the number of effected users. Whereas incomplete coverage of service issues in cloud services underestimate average level of outages, not regarding the number of effected users overestimate outages. These two biases in results are therefore opposite in direction and they partially undo each other.

Besides technical issues, there are other possible issues that could make cloud service unavailable. Cloud service provider going out of business is the most obvious and total of any such events. There is also example of case where large police investigation

closed data center for several days, because one of its customers was suspect of criminal activity. This event harmed also other customers of the same data center [3].

8.2 Estimating Internet connectivity

Important factor in cloud resources is that they are accessed through network and because examined cloud is public cloud, network in this case necessarily means the Internet. If Internet connectivity is not available, network access to cloud computing resources is not available either. Andersen D. G. [37] discusses availability of the Internet. He points out, that end-to-end Internet availability is generally in the 95-99.6% range. In the case of the Internet, there are several mechanisms for providing fault-tolerance. Not only do many hardware systems, such as routers, contain redundant components. Most importantly routing protocols are designed to compensate for link or router failures by establishing an alternate path through the network using different physical links. Despite this end-to-end availability is not perfect. Clearly if there is no route, a service cannot be available, but even the presence of a route does not guarantee the availability. Andersen lists three reasons for this. First, Internet services depend upon the proper functioning of a whole chain of components. It only takes one unmasked fault to interrupt this chain. Second, the underlying routing systems take quite a time to react with an effective route failover, disrupting connectivity. Thirdly, there are failures, which the underlying system has not been defined to react. Such failures are misconfiguration or traffic overload. Reports about Internet availability have wide spectrum. Google measurements of Internet availability indicate, that estimate of average Internet availability is less than 99.9%, when Google servers are one of the end points [38].

8.3 Public cloud failures

There is scarce information about service failures in major cloud providers. While there is little information about frequency and duration of service outages, there is even less detailed information about actual failures, information about what happened and what caused these service events. Amazon has disclosed accounts of several of its service events. Other cloud providers don't seem to publish accounts at least habitually. Microsoft did publish account of Microsoft's cloud platform, Windows Azure, service disruption on 29.2.2012, but it cannot be confirmed, that this event would have been the first service event, that Windows Azure has experienced. Case company experienced

noticeable service outage in Amazon cloud on 7.8.2011. There is no indication of service events in EU region (Ireland) since then, but same is not true for US East region (Virginia), which has experienced two outages after 7.8.2011.

Outage in EU region on 7.8.2011

Sequence of events was first triggered, when electric utility provider suffered a failure of a 110kV 10 megawatt transformer in Dublin Ireland [36]. Initially it was believed, that cause for losing the transformer was lightning, but this could not be confirmed by electric utility provider. All backup generators started normally and most generators connected online. Some portion of generators, however, did not finish connecting online because of problems in control. When situation persisted and uninterruptable power supplies (UPSs) drained, one of the three availability zones lost power to most of the EC2 instances and EBS volumes. Availability zone also lost power to networking gear. Connections from this availability zone to the Internet and to the other availability zones in the region were both lost. Power outage ended shortly, but problems persisted, because they were prolonged by several different dormant weaknesses, that were triggered by downtime. Some of the faults manifested themselves for the first time.

EC2 management service handle and divide load between availability zones inside regions. Management service however, had design flaws, which overloaded management servers with incorrect requests when large part of the instances was unable to start. Data in EBS volumes is replicated across several nodes. When large amount of nodes was offline, partially affected EBS instances caused lag by re-mirroring data, which eventually ended in storage capacity depletion inside availability zone and frozen EBS instances. When recovering the EBS instances was able to continue, recovery was prolonged by data consistency checking. This was because EBS data volumes, which had in-flight writes at the time of the power loss, had potential to be in an inconsistent state. Multi-az RDS instances have two separate database instances in different availability zones. When primary database instance is affected by disruption, election protocol elects backup copy as a new primary copy. This election protocol was disrupted by DNS connectivity issue, which downtime of networking gear triggered. Failover of multi-az RDS instances was so disabled. When DNS connectivity was restored, software bug triggered by DNS connectivity downtime caused further delay in failover. Also RDS was affected by EBS recovery problem, which was an on-going

event at the same time.

Outage in US East region on 29.6.2012

Service event started during large thunder storm, which swept through the northern Virginia area [40]. One particular data center in one of the availability zones in the US East region experienced problems with its generator power, which was unstable. Actual power outage was short, not longer than 20 minutes at most. Affected resources were numerous, EC2 instances, EBS storage volumes, RDS instances and ELB instances but no networking gear was affected. In total 7% of the EC2 instances in the US East region were impacted by the power loss. This event had some similarity with event in EU region on 7.8.2011. Again there was infrastructure failure that was caused by power failure and downtime triggered several dormant weaknesses. Similarly service event lasted significantly longer than power outage. Individual factors that prolonged the recovery process were mostly different.

Recovery of EBS volumes, which took several hours, is notable similarity between this event and service event in EU region on 7.8.2011. EBS data volumes, which have in-flight writes at the time of the power loss, have potential to be in an inconsistent state. During this event time for the completion of EC2 instance recovery was extended by a bottleneck in server booting process. The recovery of EC2 instances and EBS volumes was also prolonged by degradation of management service. EC2 management service handles launching EC2 instances and creating EBS volumes. Management service however, is distributed into several availability zones and it had design flaws, which prevented failover of primary data store of management service. Another reason for degradation of management service was caused by a flaw in ELB recovery. Failure resulting from the design flaw was triggered by a large number of ELB instances coming up after outage and it overloaded the management service with a flood of requests. Degraded ELB management delayed shifting traffic from affected availability zone to other availability zones. Recovery of RDS instances was delayed by prolonged recovery of EBS volumes, which RDS depends on. Another problem with recovery of RDS was a software fault that was triggered by the experienced sequence of communication failures. Software failure prevented some amount of multi-az RDS instances from completing failover.

Outage in US East region on 13.6.2012

Available sources about this outage are incomplete. Exact date of the outage is not certain, but it is almost certain, that this account of service failure is the first of the two outages, that US East region (Virginia) experienced during June 2012. This account is about service outage of SimpleDB database service in AWS [41]. Apparently service failure in SimpleDB was not accompanied by any large scale outages in other AWS services like EC2 instances. Nevertheless frozen SimpleDB is enough to stop any application that is dependent on SimpleDB.

Amazon has described roughly the internal structure of SimpleDB. There are three functional tiers of servers in SimpleDB. User data is stored in domains, equivalent of table in relational database. There are storage nodes, which are responsible for the storage of user data, which is replicated across several storage nodes. Second tier of servers is metadata nodes, which store metadata about user data. Example of this metadata is which storage nodes each domain is located on. There is also second layer of metadata and control, because SimpleDB is distributed database and also metadata nodes are distributed and replicated. This tier of servers controls, which nodes are responsible for a given domain. Amazon calls this tier lock service.

Failure was triggered, when multiple storage nodes became unavailable simultaneously in a single data center. Large unavailability resulted in a sudden and significant increase in load on the lock service as it rapidly removed the unavailable storage nodes from their respective replication groups. In a distributed database there is a possibility of network partition that would leave different nodes unaware of events that happen in another node, which is not in same network partition. This is why each node handshakes with the lock service periodically to verify, that it still has the responsibility for the user data or metadata, which it hosts. Overload on lock service resulted in increased latency in handshakes between healthy SimpleDB nodes and the lock service. Latency eventually exceeded handshake timeout value, which caused each healthy node to stop operating and freeze. Unravelling the fabric of distributed database continued until the whole service was unavailable.

Amazon declared that it will prevent this kind of event from re-occurring by doing rather simple adjusting to handshake protocol and its timeout value. The whole service event could be concluded to be caused by running out of capacity, despite the fact, that the failure manifested itself after significant infrastructure failure.

Conclusions from service event

Already these three events reveal information and conclusions can be done based on it. First of all, it is noticeable, that during first two service events service degradation was not isolated into primarily affected availability zone. Users of instances in other availability zones experienced problems also and processing load and communication traffic could not be shifted to unaffected availability zones. This was due to weaknesses in EC2 management system. These accounts reveal that EBS was service, which was particularly vulnerable. Recovery of EBS took a long time, according to account, that Amazon has given, many hours. EBS affects also RDS, but only in recovery. When RDS instance continues running, EBS is not used. This makes multi-az instances more resilient. Multi-az instances however, seem to be error prone, when there are communication failures. Although the third service event was mainly failure of SimpleDB, based on the two previous service events can be concluded that SimpleDB has track record of being more robust than RDS.

Chapter 9

Business continuity and service availability

9.1 Risk management

Dependence on IT has grown so big, that IT failure threatens business continuation in practically any company. This is why planning business continuation involves managing IT crises. Long service outage or significant loss of data represents potential threat to business continuation in case company. This thesis work examines possible downtime including outages in cloud service and any other cloud related incident that compromises service, which application provides. Examined threats include expected cloud related failures as well as unexpected disasters. Failure of virtual instance is an expected failure. Bankruptcy of cloud service provider and large data loss because of an administrator user error are examples of unexpected disasters. These events form a whole spectrum of incidences that have different severities. Reliability of computer hardware can be analyzed even mathematically, but more complex issues like the probability of service provider bankruptcy cannot. Different events have different probabilities and also different consequences. Risk management includes not only discovering threats and prioritizing them, but also estimating the cost of preventing each threat from damaging company.

9.2 Outages in case system

Case company has signed service level agreements with retail companies that are customers of mobile marketing service. Requirement levels in SLAs differ from customer to customer, but they range between 95% and 98.5%. Number of users of marketing service varies depending on day and time. Although consequences of possibly occurring outage vary depending on time of occurrence, service is required to

operate around the clock. Fulfilling SLA and customer satisfaction are important considerations for the case company. So far case company has been successful in fulfilling SLA. Availability of services of AWS has a significant role in fulfilling the SLAs. Luckily AWS met expectations and case company is overall satisfied with the availability of Amazon cloud.

Case company has experienced one significant lengthy period of service downtime in 2011. Primary cause of downtime was outage in EU region of Amazon. Service failure in 2011 affected all three availability zones of EU region, eu-west-1a, eu-west-1b and eu-west-1c, although outage was not complete in two of the zones. Consequence of big outage incident in 2011 for case company was complete service downtime, which lasted for 12.5 hours. Also system was not able to recover automatically after cloud service outage. The whole system needed to be manually restructured before restarting the system. Losing data would have been serious consequence from the service event, but all data was recovered by AWS and without any involvement from the case company.

Customers were not able to use the service during downtime, but in the end there were no more consequences to customers from the service event. Business of the case company was not seriously affected by the outage, partly because of the time, when outage occurred. If the incident had happened in busier time, affect to the business of the case company might have been somewhat bigger.

9.3 Using multiple cloud providers

The concern about cloud service availability calls for using multiple cloud providers as a resolution [3]. When whole system is possible to deploy to some other cloud, transferring the system to other cloud could be done as a failover, if long enough cloud service outage occurred. Using multiple cloud providers is only possible, if any kind of lock-in into one cloud provider can be avoided. That requirement is very restricting, because it makes impossible to use any proprietary API, that cloud provider might provide. In the case of designing the architecture of mobile marketing system, the case company required, that the system must be possible to deploy also to other infrastructure than the cloud that is primarily chosen for deployment. This requirement was fundamental issue in architecture. It was an argument for choosing open-source tools and APIs for the system. Choosing open-source tools favored choosing AWS as cloud provider. AWS used to be a cloud platform that was based on usage of open-

source code, whereas Windows Azure was based on proprietary licensed software. Open-source programs however, are not distinctively characteristic for AWS platform anymore. Difference compared to Windows Azure is not crisp, both because AWS offers now Windows virtual machines and Windows Azure platform offers Linux virtual machines. Amazon anyway became chosen as cloud provider and a set of cross-platform open-source languages, utilities and tools became to form the framework of the system. Server side implementation utilized Java language, which is cross-platform language. Several open-source Java libraries were used, Spring framework, Java Persistence API and Vaadin web UI framework. All used libraries were platform independent.

Developing system to be platform independent was successful. The whole mobile marketing system was deployed to secondary cloud provider's platform as a test to verify the cross-platform implementation. Test was done before the system eventually first went live. Case company has not created detailed plans for transferring the system to other cloud although there is ability to do it.

Long cloud service outages can be estimated to be rare. ARWC report lists two incidents, where cloud service outage has lasted approximately one week. First such incident happened in 2007, when NaviSite was down for one week for an unknown reason. Second incident happened in 2009, when OVH was down also for one week because of a file system failure. ARWC report does not record any outage longer than one day that Amazon has experienced. Probability for one week cloud service outage is very low based on statistics. Also it seems highly unlikely, that Amazon web services would go out of business any time soon. Amazon is indisputable market leader in IaaS cloud and risks would be different in case of some other cloud provider.

Transferring complete system from one cloud to another requires transferring also the data that the system uses. Exporting data is easy, when used database is not proprietary to used cloud provider. Open-source database MySQL is a good example of such non-proprietary database. RDS instances expose interface that is equal to native database engine irrespective of chosen database engine. Transferring the data is easier, when it happens premeditated rather than as a consequence of unexpected crisis or outage. If there is outage of RDS for example going on, up-to-date data cannot be exported. If data, which has been exported from cloud database as a data backup already before the outage started, is used, those updates to database, that have been done after taking

backup snapshot, are lost. It is noticeable, that data, that in AWS has been stored in EU region, is not available in other regions unless it has been copied into S3 storage service for example.

One possibility for using several cloud providers or regions is to create hot standby system, which is a replica of live system. Only one system would be used but data would be replicated in nearly real-time to hot standby replica. This would make failover from one cloud to another possible in considerably short time and with little data loss. Probabilities of risks need to be considered in prevention and probabilities of long service outage and service provider going out of business have considerably low probability. Because hot standby would add significant costs to the system, it is not noteworthy to study it in detail in this case study.

9.4 Managing IT crisis in case company

Besides service availability also data integrity and data durability are important characteristics of a dependable system. Customers are unlikely to consider system that loses or corrupts customer's data, as dependable. Amazon's track record is spotless in keeping case company's data safe. There is no single known issue of lost or corrupted data during the time that Amazon has provided cloud services for the case company. Data in EBS is stored redundantly, though failure and loss of data is possible in EBS unless backed up with snapshots. S3 on the other hand has been designed to be ultra-durable. Case company has not experienced loss or inconsistency of data in RDS either.

Focusing only on technical aspect of data durability leaves dangerously room for an error. At least when there is some redundancy and fault-tolerance in data storage, user remains the biggest threat to keeping data. Such techniques as RAID, that protect data from technical failure, do not help when user deletes or corrupts data. This is why storage technology needs to be complemented by taking regularly backups. Administrator user might damage system accidentally also other ways. Case company found, that there are few definite means to prevent administrator user to damage system. There are precautionary measures however, that are feasible, like taking backups and storing them outside the cloud and writing guidelines, how to perform tasks.

During the deployment project of mobile marketing system case company made only rough plans, how to handle possible IT crises related to mobile marketing system. No detailed plan for crisis management was ever recorded or drilled by personnel.

Chapter 10

Discussion and conclusions

10.1 Discussion

10.1.1 Cloud service availability

Cloud computing as a computing model still provokes uncertainty. Availability of cloud service needs assurance and customers look for that assurance in reputation, history and service level agreement. In current phase of rapid growth, even market leader that is best known among its own type of cloud providers has rather short history. Solid statistics about cloud providers are still scarce.

Disclosures of service events in Amazon web services reveal that while they were triggered by large scale infrastructure failures, they were made worse by software faults and design faults in AWS. Mayhem was made worse by cascading failures of management services and other services. Problems, which the case company experienced during one service event, can be explained with these situations. Under those circumstances there were unexplained failures and phenomena. In normal situation, however, there are few failures and they happen in a controlled way. Based on experiences of the case company, IaaS cloud is manageable. Cloud users would like to have more visibility to cloud, but with enough knowledge they cope with managing cloud.

Network access is distinctively characteristic to cloud computing. When cloud is located outside of local network of company, used network is Internet. Any cloud that is used by consumers is accessed through Internet irrespective of the type of client that is used. Cloud client might be in desktop platform or increasingly also in mobile platform.

Internet availability limits also service availability and because internet availability is relatively low, Internet availability is rather predominant to users' perception of service availability. Even brief interruptions lasting more than a few seconds can degrade users' perception of a site's performance and such interruptions in Internet connectivity are frequent. It is long outages lasting many hours or more, which are distinctive and degrade public perception of a service. If service in question is big and well known, outage also gains public attention and news coverage, although there are not many sites in the Internet, which are prominent enough to catch such attention. Some individual Internet users have been observant enough to report, that when Amazon is experiencing downtime, "half of the Internet disappears".

Several outages, that Amazon and other cloud providers have experienced, have attracted much attention. One stated assumption about the cause of many outages is rapid growth of the demand of services, which allegedly had outpaced actual computing resources. Evidence from Amazon tells differently. Possibly the problems, that have been seen in dependability of cloud, will be fixed in the future. For example Amazon started EC2 offering without any SLA and now provides guarantee of 99.95% availability. Another example is OVH, which experienced notable outage in 2009, but now offers 99.99% SLA. Even if OVH somewhat slips from its promise, it still is in sharp contrast with past.

Notable about examined system is that it is Internet service. It is publicly available in Internet and used mainly with mobile phones. This is significant distinction to any system that is internal system in enterprise. When system does not have such requirements for high-availability, that using undependable public Internet would not be tolerable, using public cloud is tolerable also. Region 'EU' of AWS for example resides in Ireland. Conclusion would be opposite, if system did not tolerate that network distance. According to available information none of the big cloud providers has such bad reputation, that it could not be given a recommendation. Case company runs a service, that somewhat tolerates latency and downtime. Requirements that were initially set for the system were met easily and there is no need to withdraw from cloud.

10.1.2 Elements in cloud supporting availability

Designer of a web service has available building blocks that are inherent for web application. Basic building blocks of web service might be taken as granted nowadays

although they were carefully crafted at the same time as HTTP. RESTful web application adheres to constraints that were created same time as web initially was designed. This is why it is real irony that in typical contemporary jargon REST is labeled as so called web 2.0. Examined system has multi-channel access. RESTful interfaces are a characterizing pattern in the examined system, which has web and mobile clients. Examined system has been seen to have high availability and also other appropriate architectural properties.

Google's PaaS cloud platform is an example of a platform that provides automatic fault-tolerance mechanisms. Whereas such platform provides high availability conveniently, it also forces application into a strict web application programming model. When cloud service is built on IaaS cloud, platform does not provide automatic fault-tolerance mechanisms because it is impossible in unrestricted IaaS platform. Building fault-tolerance is then the responsibility of architect, because fault-tolerance is application dependent.

Case system used several managed services, that cloud provider offered. At least one of them, namely ELB, was found to lack some features that licensed software would have had. Nevertheless managed services are obviously attractive. Some of them have proprietary APIs, which constitutes a risk of lock-in. One possibility is to use third-party software to create standard interface instead of proprietary API. This is possible for SimpleDB for example, which can be used with Java Persistence API on top of it. Those who do not tolerate the risks, that proprietary software services or APIs bring, also lose the benefits that they offer. SimpleDB is resilient to communication failures because it is a distributed datastore and eventually consistent. It seems very likely, that SimpleDB would prove to be even more robust than RDS.

10.1.3 Application architecture

Examined system proved to be success. Its architectural design created high availability and a system that is easy to administer. There seems to be no need for improvement of architecture. Although examined system seems to be sufficient as it is, AWS has some opportunities that are currently unused. When examined from the point of view of availability, RDS is sufficiently resilient. Nevertheless SimpleDB or Dynamo, which are more native to cloud computing, might provide highly available database in future projects, which utilize AWS. SimpleDB has also other benefits like easiness of

management. There are new software services introduced frequently by different cloud providers and they present an opportunity for anyone who makes use of them.

Examined design differs from model architecture, which Amazon has provided for developers. It can be seen, that this is due to small size of examined system. This does not mean that the system would not be scalable. Growing its size would increase availability of the system, because redundancy would grow.

10.1.4 Planning for recovery in cloud

Distinctively characteristic for actual cloud is that it has API, which can be used for managing cloud resources. This makes possible to automate tasks in administering cloud. Monitoring tools, management API and automating tasks makes possible to achieve shorter time to repair, when there is some kind of failure in system. Shorter time-to-repair transfers to bigger time portion of system availability. Automation also has potential to help administering systems by helping doing more tasks and procedures planned instead of coming up with ad hoc solutions. Increasing planning helps to make improving system availability into more like a continuous process. Especially compared to real on-premise hardware, cloud computing can make planning for high availability, if not simpler, at least more clear.

10.1.5 Evaluation of results

This empirical enquiry includes many different aspects. Among all discussed issues it is most difficult to get evidence about cloud service dependability. Currently cloud services is not completely established sector of services, but un-established is not the same thing as undependable. Cloud services lack visibility and there are only few accurate sources of evidence about dependability and outages. Considering the goals of this study, however, enough evidence was acquired. Essential evidence in this study related to this matter is interview of technology manager in case company. After comparing different sources like interviews, SLA documents and online documents there is no significant discrepancy between sources.

Scope of this study includes examining in detail only the services and applications of cloud service provider, which the case company used for examined system. This particular IaaS cloud and its accompanying services are well documented and also external sources about it exist. Documentation and external sources provide enough

information about how infrastructure services and software services of this particular cloud provider support availability of examined system. Also there is sufficient information for making contingency planning of this cloud deployed system possible.

Architecture documentation of examined system was accurate enough for the goals of this study. Accuracy of architecture documentation was confirmed by the architect who designed the system. Self-service characterizes cloud services in general and this is great help, because many essential elements of architecture are standardized and contain only few user specific configurations. This is true for both used infrastructure and software that is acquired as a service. This makes overall architecture simpler, clearer and adds visibility. Documentation of reference architecture provided by cloud service provider was also helpful for increasing insight.

Service that was examined is rather small and examined company that operates it operates like small company operates. There are less defined procedures for operation and less documentation than in some larger company. This makes harder to examine planning in company, because planning lacks detail and written documents. Reaching involved people for interview was an obstacle for finding requirements of the system, experiences of its use and business perspectives, as well as for finding out some technical details of examined system. Only two people in different positions were interviewed. Examined system is not very big and there are only few people involved with it. Even if enquiry had been more extensive, there would not have been many people to interview. Results from interviews do not show any significant discrepancy, but small number of interviews makes it difficult to find differences in opinions. Tone in interviewing both persons was overall positive, which also can be presumed to moderate any possible differences. Undersigned has positive attitude towards cloud computing in general, but this does not have influence on results.

Scope of this study includes examining one Internet service, its implementation and deployment into IaaS cloud. Context of the study is this particular application and this particular IaaS cloud. Many results of this study are not valid in some other context. Choice of provider of IaaS cloud is substantive issue in cloud deployed system. IaaS cloud, which is provided by some particular service provider, contains fault-tolerance mechanisms that are similar to PaaS environment. Some other IaaS cloud contains more or less fault-tolerant hardware. Each infrastructure service differs in resilience and

recovery. Also management and monitoring are different in different cloud providers. Notable characteristics of Amazon web services are software services that are managed by service provider. No other IaaS cloud offering contains the same software services. Although RDS for example is available with MySQL database engine, which is open-source product, RDS has special characteristics, which differentiate it from any available database product or software service offering.

Discovering requirements for availability and failure and disaster recovery in IaaS cloud context was one of the central goals of this study. These requirements are highly dependent on the application in question and only some of them can be generalized to larger spectrum of Internet services. Examined system can be compared to other Internet services, but requirements for availability vary from application to application.

This particular service is rather small and size is an essential factor in architecture of an Internet service. Spectrum of the size of currently existing Internet services is significant. Biggest services have Internet scale. Such super-giant size services have completely different architecture compared to system, which was examined. Curiously Amazon cloud brings some of the architectural elements available to anyone as software services. Still scale is determinative of generalizing results, which are dependent on application architecture. There are however discovered phenomena, which are generalizable to all sizes of Internet services. One is that user of system is significant threat to the resiliency and availability of system. This has implications to disaster and failure planning. One central observation is that end-to-end Internet availability is determinative of the perceived availability of Internet service. Internet services with different scale most likely have similarity in this respect.

10.2 Conclusions

Deploying software system into IaaS cloud takes infrastructure out of user's control. Acquiring infrastructure as a service diminishes visibility into infrastructure and changes system administration. Service outages of infrastructure services and other risks to availability have caused concern for early users of cloud. The purpose of this master's thesis was to evaluate existing web application, which is deployed in IaaS cloud, for availability. Examined threats to availability form a spectrum of incidences that have different severities.

Conventional mean to increase availability of software system is to create redundancy into system. Cloud service providers build mutually independent domains or zones into infrastructure to better support availability.

Big service providers have currently service level agreements effective. Data about historical availability of several service providers shows that longer term availability of random cloud service provider that is below 99.9% is clearly exceptional. Long cloud outages are rare events. Cloud resources are accessed through network and public cloud availability is dependent on end-to-end Internet connectivity, which is relatively poor. Internet service, which was evaluated, has good enough availability that Internet end-to-end availability is determinative of users' perceived performance of site.

Case company has experienced one significant considerably lengthy period of cloud service outage, but is overall satisfied with the availability of cloud service. There is no need to withdraw from public cloud because of outages or any other reason either. Case company runs a service, that somewhat tolerates latency and downtime.

One recognized risk is that service provider goes out of business or there is some other reason to change service provider. Using multiple service providers is a solution to cloud service unavailability. This requires independence from service provider and that was decided to be a requirement for system architecture. Case company planned and tested migrating Internet service to cloud of another service provider as a failover.

User is a significant threat to the resiliency of system, but there are no definite means to prevent user from damaging system. Case company created plans how to handle IT crisis and taking routinely and regularly backups of data outside the cloud is the core action in IT crisis preparedness. Case company has not experienced any actual IT crisis after the Internet service went live.

Architecture of the examined system was discovered to fulfill set requirements and no actual suggestions for improvements were found. Examined system has application architecture, which is conventional in web applications. It has stateless servers in application tier. Resiliency to instance and networking failures is supported by managed database service and managed load balancer, which are advanced features from IaaS provider. Both elements of architecture are inherent for web application and used together with the pattern of RESTful interfaces in application architecture.

System is made repairable with stateless recovery, which is conceptually simple. Recovery of failed virtual machines is automated in examined system, although it requires administrator involvement.

References

1 Yin, R. K. Case study research: design and methods. 3rd ed. Sage Publications, 2003. 181 p. ISBN 0-7619-2552-X.

2 Mell, P. & Grance, T. The NIST Definition of Cloud Computing. [Online]. National Institute of Standards and Technology (NIST), U.S Department of Commerce. 2011. [Referenced 18.8.2012]. Available: http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf

3 Armbrust, M. & Fox, A. & Griffith, R. & Joseph, A.D. & Katz, R.H. & Konwinski, A. & Lee, G. & Patterson, D.A. & Rabkin, A. & Stoica, I. & Zaharia, M. A View of Cloud Computing. Communications of the ACM. [Online magazine]. Vol. 53:4. 2010. P. 50-58. [Referenced 18.8.2012]. ISSN 0001-0782. DOI:10.1145/1721654.1721672.

4 Leavitt, N. Is Cloud Computing Really Ready for Prime Time? Computer. [Online magazine]. Vol. 42:1. 2009. P. 15-20. [Referenced 18.8.2012]. ISSN 0018-9162. DOI:10.1109/MC.2009.20

5 The Open Group Cloud Computing Survey. [Online]. The Open Group. May 2011. [Referenced 18.8.2012]. Catalog number R110. Available: http://www.opengroup.org/cloudcomputing/uploads/40/24144/Open_Group_Cloud_Computing_Survey_FINAL.pdf

6 Armbrust, M. & Fox, A. & Griffith, R. & Joseph, A.D. & Katz, R.H. & Konwinski, A. & Lee, G. & Patterson, D.A. & Rabkin, A. & Stoica, I. & Zaharia, M. Above the Clouds: A Berkeley View of Cloud Computing. [Online]. University of California at Berkeley. 10.2.2009. [Referenced 18.8.2012]. Available: http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.html

7 Rimal, B.P. & Choi, E. & Lumb, I. A Taxonomy and Survey of Cloud Computing Systems. In: Fifth International Joint Conference on INC, IMS and IDC, 2009. NCM '09. Seoul, 25.-27.8.2009. IEEE Computer Society, 2009. P. 44-51. ISBN 978-1-4244-5209-5. DOI:10.1109/NCM.2009.218.

8 Jiménez-Peris, R. & Patiño-Martinez, M. & Kemme, B. & Alonso, G. Improving the Scalability of Fault-Tolerant Database Clusters. Proceedings of the 22nd International Conference on Distributed Computing Systems, 2002. IEEE Computer Society, 2002. P. 477-484. ISBN 0-7695-1585-1. ISSN 1063-6927. DOI:10.1109/ICDCS.2002.1022297.

9 Hacker, J. K. & Ahson, S. A. (ed.) & Ilyas, M. (ed.). Cloud Computing and Software Services : Theory and Techniques. Boca Raton, Florida, USA: CRC Press, 2011. 442 p. ISBN 978-1-4398-0315-8.

REFERENCES

10 Avizienis, A. & Laprie, J.-C. & Randell, B. & Landwehr, C. Basic Concepts and Taxonomy of Dependable and Secure Computing. IEEE Transactions on Dependable and Secure Computing. [Online journal]. Vol. 1:1. 2004. P. 11–33. [Referenced 18.8.2012]. ISSN 1545-5971. DOI:10.1109/TDSC.2004.2.

11 Lyu, M. R. & Mendirratta, V. B. Software Fault Tolerance in a Clustered Architecture: Techniques and Reliability Modeling. In: Proceedings of the 1999 IEEE Aerospace Conference, 1999. Snowmass at Aspen, CO, USA. 7.3.1999. IEEE Computer Society, 1999. Vol. 5. P. 141-150. ISBN 0-7803-5425-7. DOI:10.1109/AERO.1999.790197.

12 Torres-Pomales, W. Software Fault Tolerance: A Tutorial. [Online]. NASA Langley Research Center, Hampton, Virginia, USA. NASA Langley Technical Report Server. 2000. [Referenced 18.8.2012]. Technical Memorandum TM-2000-210616. Available: http://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20000120144_2000175863.pdf

13 Van Vliet, J. & Paganelli F. Programming Amazon EC2. 1st ed. O'Reilly, 2011. 163 p. ISBN 978-1-449-39368-7.

14 Reference Model for Service Oriented Architecture 1.0. [Online]. OASIS. 2.10.2006. [Referenced 22.8.2012]. Document identifier: soa-rm. Available: http://docs.oasis-open.org/soa-rm/v1.0/

15 Reese G. Cloud Application Architectures. 1st ed. O'Reilly, 2009. 184 p. ISBN 978-0-596-15636-7.

16 Oppenheimer, D. Ganapathi, A. Patterson, D. A. Why do Internet services fail, and what can be done about it? In: Proceedings of USITS'03: the 4th conference on USENIX Symposium on Internet Technologies and Systems - Volume 4. Seattle, WA, USA. 26-28.3.2003. CA, USA: USENIX Association Berkeley, 2003. P. 1-16. Available: http://static.usenix.org/events/usits03/tech/full_papers/oppenheimer/oppenheimer.pdf

17 Fielding, R. T. & Taylor, R.N. Principled design of the modern web architecture. ACM Transactions of Internet Technology (TOIT). [Online journal]. Vol. 2:2. May 2002. P. 115–150. [Referenced 18.8.2012]. DOI:10.1145/514183.514185. ISSN 1533-5399.

18 Fielding, R.T. REST APIs must be hypertext-driven. Untangled, musings of Roy T. Fielding. [Online]. Fielding, R.T. 20.10.2008. [Referenced 18.8.2012]. Available: http://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven

19 Carolan, J. & Gaede, S. & Baty, J. & Brunette, G. & Licht, A. & Remmell, J. & Tucker, L. & Weise, J. Introduction to Cloud Computing architecture. [Online]. Sun Microsystems. June 2009. [Referenced 18.8.2012]. Available: http://www.gtsi.com/eblast/corporate/cn/09_09_2009/PDFs/Sun.pdf

20 Girdlay, M. & Woollen, R. & Emerson, S. L. J2EE Applications and BEA WebLogic Server. Prentice Hall, 2002. 664 p. ISBN 0-13-091111-9.

21 Three-tiered Distribution. Microsoft Developer Network. [Online]. Microsoft. 2012. [Referenced 18.8.2012]. Available: http://msdn.microsoft.com/en-us/library/ff647546

22 Leff, A. & Rayfield, J.T. Web-Application Development Using the Model/View/Controller Design Pattern. In: Proceedings of the Fifth IEEE International Enterprise Distributed Object Computing Conference, 2001. EDOC '01. Seattle, WA, USA. 4.-7.9.2001. IEEE Computer Society, 2001. P. 118–127. ISBN 0-7695-1345-X. DOI:10.1109/EDOC.2001.950428.

23 Deployment Patterns. Microsoft Developer Network. [Online]. Microsoft. 2012. [Referenced 18.8.2012]. Available: http://msdn.microsoft.com/en-us/library/ff646997

24 Elmasri, R. & Navathe, S. B. Fundamentals of Database Systems. 3rd ed. Addison-Wesley, 2000. 955 p. ISBN 0-8053-1755-4.

25 Özsu, M. T. & Valduriez, P. Principles of Distributed Database Systems. 3rd ed. Springer, 2011. 845 p. ISBN 978-1-4419-8833-1.

26 Gilbert, S. & Lynch, N. Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-Tolerant Web Services. ACM SIGACT News. [Online newsletter]. Vol. 33:2. June 2002. P. 51-59. [Referenced 18.8.2012]. ISSN 0163-5700. DOI:10.1145/564585.564601.

27 Vogels, W. Eventually Consistent. Communications of the ACM. [Online magazine]. Vol. 52:1. January 2009. P. 40-44. [Referenced 18.8.2012]. ISSN 0001-0782. DOI:10.1145/1435417.1435432.

28 Cloud Computing: Current Market Trends and Future Opportunities. [Online publication]. Cloud Times. 22.6.2011. [Referenced 18.8.2012]. Available: http://cloudtimes.org/2011/06/22/cloud-computing-its-current-market-trends-and-future-opportunities/

29 Amazon Elastic Compute Cloud (Amazon EC2). [Online]. Amazon. [Referenced 21.8.2012]. Available: http://aws.amazon.com/ec2/

30 Elastic Load Balancing. [Online]. Amazon. [Referenced 21.8.2012]. Available: http://aws.amazon.com/elasticloadbalancing/

31 Amazon S3 Service Level Agreement. [Online]. Amazon. [Referenced 21.8.2012]. Available: http://aws.amazon.com/s3-sla/

32 Monitoring Instances with Status Checks. [Online]. Amazon. [Referenced 2.10.2012]. Available: http://docs.amazonwebservices.com/AWSEC2/latest/UserGuide/monitoring-system-instance-status-check.html

33 Amazon EC2 Service Level Agreement. [Online]. Amazon. 23.10.2008. [Referenced 15.10.2012]. Available: http://aws.amazon.com/ec2-sla/

34 Windows Azure: Service Level Agreements. [Online]. Microsoft. [Referenced 15.10.2012]. Available: http://www.windowsazure.com/en-us/support/legal/sla/

35 OVH Cloud: Security and guarantees. [Online]. OVH. [Referenced 15.10.2012]. Available: http://www.ovh.co.uk/cloud/why_ovh/security_and_warranty.xml

36 Paris University and Telecom ParisTech Publish Availability Ranking of World Cloud Computing. [Online]. International Working Group on Cloud Computing Resiliency. [Referenced 2.10.2012]. Available: http://iwgcr.org/paris-university-and-telecom-paristech-publish-availability-ranking-of-world-cloud-computing

37 Andersen, D. G. Improving End-to-End Availability Using Overlay Networks. [Online]. Massachusetts Institute of Technology (dissertation). 2005. [Referenced 2.10.2012]. Available: http://angio.net/res/andersen-phd.pdf

38 Barroso, L. A. & Hölzle, U. The Datacenter as a Computer : An Introduction to the design of Warehouse-Scale Machines. Morgan & Claypool, 2009. 105 p. ISBN 9781598295573. DOI:10.2200/S00193ED1V01Y200905CAC006.

39 Summary of the Amazon EC2, Amazon EBS, and Amazon RDS Service Event in the EU West Region. [Online]. Amazon. [Referenced 2.10.2012]. Available: http://aws.amazon.com/message/2329B7/

40 Summary of the AWS Service Event in the US East Region. [Online]. Amazon. 2.7.2012. [Referenced 2.10.2012]. Available: http://aws.amazon.com/message/67457/

41 Summary of the Amazon SimpleDB Service Disruption. [Online]. Amazon. [Referenced 2.10.2012]. Available: http://aws.amazon.com/message/65649/

Appendix A

A Interview questions

RELIABILITY OF INSTANCES

1a. Has there been failures of EC2 instances?

1b. If yes, did it have noticeable impact to service of your company

FAILURE MODE OF INSTANCES

2a. Has there been failures of EC2 instances, that did not recover normally? Example of such incident could be, that you can see from management console, that instance is running, but you cannot connect to it.

2b. If yes, did you contact Amazon customer service or developer forum?

RELIABILITY OF SERVICES

3a. Has there been failures of other AWS services like RDS?

3b. If yes, did it have noticeable impact to service of your company

AVAILABILITY OF CLOUD

4a. Has there been any significant outages in cloud services?

4b. If yes, what kind of effect they have had to your business?

INTERVIEW QUESTIONS

OTHER DEPENDABILITY

5a. Has there been any other significant disruptions than outages of cloud services? Example of such disruption could be lost or corrupted data.

5b. If yes, was it caused by Amazon services, user mistake or something else?

MEETING REQUIREMENTS FOR AVAILABILITY

6a. Has system met the overall requirements, that were set for its availability?

6b. If not, what kind of effect has it had to your business?

ADMINISTERING

7a. Has administering production system proved easy and economical?

7b. If not, what has been the obstacle in administering production system?

BUSINESS CONTINUATION PLANNING

8a. Did you anticipate and assess threat scenario, that there would be long term (at least several days) outage in cloud service?

8b. Did you anticipate and assess threat scenario, that cloud provider goes out of business?

8c. Did you anticipate and assess threat scenario, that administrator user makes mistake which breaks production environment?

8d. Did you anticipate and assess threat scenario, that would happen some other failure not mentioned above?

CRISIS MANAGEMENT

9a. Did you make in advance any plans for survival in case of a catastrophic failure of the system?

INTERVIEW QUESTIONS

ANTICIPATION OF OUTAGES

10a. How did you anticipate outages in the system before implementation was done?

10b. If you did expected outages to happen, what did you estimate that the effect of expected outages to your business would be?